(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 717 353 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
05.02.1997 Bulletin 1997/06

(43) Date of publication A2:
19.06.1996 Bulletin 1996/25

(21) Application number: **95308717.8**

(22) Date of filing: **01.12.1995**

(51) Int. Cl.[6]: **G06F 9/44**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **14.12.1994 US 355889**

(71) Applicant: **AT&T Corp.**
New York, NY 10013-2412 (US)

(72) Inventors:
- **Korn, David Gerard**
  New York, New York 10003 (US)
- **Vo, Kiem-Phong**
  Berkeley Heights, New Jersey 07922 (US)

(74) Representative: **Watts, Christopher Malcolm Kelway, Dr. et al**
Lucent Technologies (UK) Ltd,
5 Mornington Road
Woodford Green Essex, IG8 0TU (GB)

(54) Efficient and secure update of software and data

(57) The invention concerns apparatus for updating a data file, from an earlier version to a later version. The invention compares the earlier version with a later version, and derives a transformation, which contains information as to the similarities and differences. The invention then processes the earlier version, using the transformation, in order to derive the later version, without reference to the later version itself.

The invention allows multiple versions of an original file, such as bank records located at multiple locations, to be updated by transmitting the transformation to the multiple locations, instead of transmitting the updated files themselves. The procedure is highly resistant to interception of the updated data, because the transformation, in general, does not contain the entire contents of the later versions.

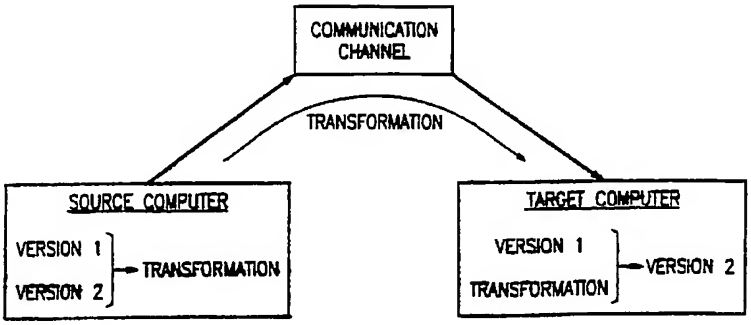


FIG. 1

EP 0 717 353 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8717

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	FR-A-2 701 777 (NIPPON ELECTRIC CO) 26 August 1994 * page 3, line 16 - page 4, line 23 *	1-4,7	G06F9/44
Y		5,6,8	
Y	WO-A-92 22870 (ICL DATA AB) 23 December 1992 * page 7, line 4 - line 36 * * page 17, line 16 - page 18, line 23 *	5,6,8	
A	WO-A-94 27219 (APPLE COMPUTER) 24 November 1994 * page 5, line 12 - page 6, line 5 *	1-8	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	4 December 1996	Brandt, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平8－255104

(43)公開日　平成８年(1996)10月１日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	12/00	５１７	7623－5B	Ｇ０６Ｆ	12/00	５１７
		５１０	7623－5B			５１０Ｂ
	9/06	４１０			9/06	４１０Ｐ

審査請求　未請求　請求項の数８　ＯＬ　（全 38 頁）

(21)出願番号　特願平7－325290

(22)出願日　平成７年(1995)12月14日

(31)優先権主張番号　０８／３５５８８９

(32)優先日　1994年12月14日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390035493
エイ・ティ・アンド・ティ・コーポレーション
ＡＴ&Ｔ　ＣＯＲＰ.
アメリカ合衆国　10013－2412　ニューヨーク　ニューヨーク　アヴェニュー　オブ　ジ　アメリカズ　32

(72)発明者　ディヴィッド　ジェラルド　コーン
アメリカ合衆国　10003　ニューヨーク，ニューヨーク，エー107，マーサー　ストリート　303

(74)代理人　弁理士　岡部　正夫　（外２名）

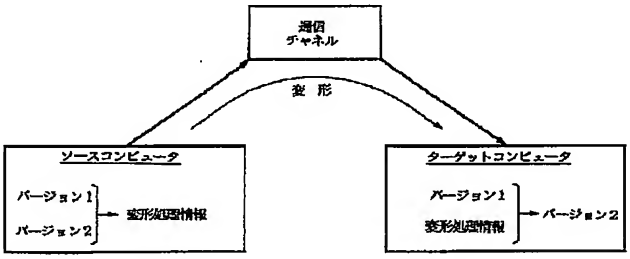
最終頁に続く

(54)【発明の名称】　ソフトウエアおよびデータの効率的かつ安全性の高い更新

(57)【要約】

【課題】　伝送データ量を削減し、かつかつ傍受されにくい、遠隔地からソフトウエア又はデータを更新する方法を提供する。

【解決手段】　本発明は、旧バージョンから新規バージョンへデータファイルを更新する装置に関する。本発明は、旧バージョンと新規バージョンを比較し、類似と相違に関する情報を含む変形処理情報を導き出す。さらに、本発明は、新規バージョンを導き出すために、新規バージョン自体を参照せずに、変形処理情報を用いて、旧バージョンの処理を行う。本発明を用いると、更新されたファイル自体を送信しなくても、変形処理情報を複数の場所に送信することにより、複数の場所にある銀行の記録などのような元のファイルの複数バージョンを更新することができる。一般に、変形処理情報には新規バージョンの内容全体が含まれていないことから、この方法は傍受に対して高い抗力を持つ。



1

【特許請求の範囲】
【請求項１】　コンピュータにおいて、
ａ）第二バージョンにアクセスしなくても、第一バージョンに基づいてバイナリファイルの第二バージョンの復元を可能にする命令を作成するプログラム手段から成る改良。
【請求項２】ａ）連結されたときに第二バージョンを復元する文字列を各々が作成する一連の命令を生成する段階と、
ｂ）各命令を実行する段階とから成る、第一バージョンから導き出されたバイナリファイルの第二バージョンを復元する方法。
【請求項３】デジタルコンピュータの場合、
ａ）
ｉ）第一および第二ファイルを検査し、
ｉｉ）第二ファイルを復元するため、実行されたときに、
Ａ）第一ファイルの一部と、
Ｂ）第二ファイルへのアクセスを行わずに第二ファイルの一部とを結合する一連の命令を作成する第一プログラム手段から成る改良。
【請求項４】　第二ファイルの復元を行うために一連の命令を実行する第二プログラム手段を具備する請求項３に記載の改良。
【請求項５】ａ）新規バージョンと旧バージョンとを比較し、かつ、
ｉ）新規バージョンが旧バージョンと似ている類似語句と、
ｉｉ）新規バージョンが旧バージョンと異なる相違語句を識別し、
ｂ）旧バージョンに各類似語句が出現しているアドレスを記憶し、
ｃ）新規バージョンに出現しているアドレスと共に各相違語句を記憶する段階とから成る、コンピュータファイルの新規バージョンを退避する方法。
【請求項６】ａ）第一サイトにおいて、第一ファイルと第二ファイルの
ｉ）類似と、
ｉｉ）相違
を検出する段階と、
ｂ）第二サイトにおいて、第一ファイルのコピーを保持する段階と、
ｃ）
ｉ）類似の位置と、
ｉｉ）相違自体と、
ｉｉｉ）（ｃ）（ｉ）、（ｃ）（ｉｉ）、および第二ファイルに基づく第二ファイルの復元を可能にする情報を、第二サイトに送信する段階とから成る情報の復元法。
【請求項７】ａ）

2

ｉ）新規バージョンの指定位置に旧バージョンの指定部分をコピーすることと、
ｉｉ）新規バージョンの指定位置に新規バージョンの指定部分をコピーすることと、
ｉｉｉ）新規バージョンの指定位置に指定バイトを加えることのうち、１つまたはそれ以上を行うよう指示する命令を受信する段階と、
ｂ）命令を実行する段階とから成る、ファイルの旧バージョンを新規バージョンへ更新する方法。
【請求項８】ａ）
ｉ）
Ａ）第一バージョンまたは第二バージョンの発信元位置から、
Ｂ）第二バージョンの宛先位置へ、文字がコピーされるよう指示するＣＯＰＹ命令と、
ｉｉ）指定文字が第二バージョンの指定位置に加えられるよう指示するＡＤＤ命令とから成る、並びを導き出す手段から成るファイルの第一バージョンと第二バージョンの相違を表すデータを作成するシステム。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明により用いられるソースコードを記載した付録を添付する。本発明は、伝送されるデータ量を削減し、かつ傍受されにくい、遠隔地からソフトウエアまたはデータもしくは両者を更新する方法に関する。
【０００２】
【従来の技術】電信電話会社の電話システムを介したコンピュータ間の電子データ通信には、スピードと機密保護が必要とされている。伝送前にデータを圧縮することにより伝送速度を向上できる一方で、暗号化により機密保護が可能である。データ圧縮は、データ源における冗長度を低下させ、かつ暗号解読への抗力を高めることから、暗号化にも有効な働きをする。しかし、データファイルが圧縮ならびに暗号化された場合、圧縮および暗号化されたデータは、単一のデータ源に基づくものとなる。単一のデータ源に基づくデータ圧縮および暗号化は、完全なものとはいえない。データの冗長度が高くなければ、圧縮はうまく機能せず、最新の暗号解読法と共に機能する高速コンピュータは、大抵の暗号化データを解読することができるからである。多くの場合、最初のデータ源に関する複数の新規バージョンが、データに若干の変更が加えられた後に頻繁に伝送される。この場合の変更が若干であるとは、全バージョンが本質的に類似していることを意味している。本質的に類似していれば、その類似性を利用して、あるバージョンを別のバージョンに変形する最小の変形処理情報を計算することが可能である。
【０００３】このような変形処理情報の計算方法は、データディファレンシング（データ間の差の計算）と呼ば

れている。データディファレンシングは伝達されるデータ量を削減する働きをすることから、これにより伝送速度が向上する。さらに、旧バージョンを持たない傍受者は、伝送されたデータからあまり多くの情報を引き出すことができないため、データディファレンシングは、プライバシーの保護にも役立つ。これまで数多くの圧縮およびディファレンシング技術が、検討されてきた。「情報理論に関するＩＥＥＥトランザクション（ＩＥＥＥ Ｔｒａｎｓａｃｔｉｏｎｓ ｏｎ Ｉｎｆｏｒｍａｔｉｏｎ Ｔｈｅｏｒｙ）」１９７７年５月号の２３（３）（３３７頁～３４３頁）に記載されたＪ．Ｚｉｖならびに A．Ｌｅｍｐｅｌによる「順次データ圧縮のための普遍アルゴリズム（Ａ Ｕｎｉｖｅｒｓａｌ Ａｌｇｏｒｉｔｈｍ ｆｏｒ Ｓｅｑｕｅｎｔｉａｌ Ｄａｔａ Ｃｏｍｐｒｅｓｓｉｏｎ）」の記事では、単一のデータファイルを圧縮する技術について述べられている。この技術は、データの順序を分析し、かつ、可能であれば、各位置においてすでに分析された部分にある別のセグメントと一致する最長セグメントを識別することによって機能する。このようなセグメントが検出された場合、一致した位置と一致した長さを用いて暗号化が行われる。また、一致しなかったデータは、そのまま出力される。

【０００４】このアルゴリズムの具体化が、１９８４年８月７日にW．Ｌ． Ｅａｓｔｍａｎ、Ａ． Ｌｅｍｐｅｌ、およびＪ．Ｚｉｖ．に付与された米国特許４，４６４，６５０の主題であった。このＥａｓｔｍａｎ－Ｌｅｍｐｅｌ－Ｚｉｖ圧縮方法では、圧縮解除中も圧縮中とほぼ同じ作業量をこなさなければならないことから、圧縮解除の際に低速となる。さらに、この方法は、データディファレンシングに応用できない。ＵＮＩＸシステムでは、第一ファイルから第二ファイルへの変形の際に削除または追加する必要のある一連の行を作成する「ｄｉｆｆ」プログラムを用いて、（バイナリ以外の）テキストファイル間の違いを算出することができる。「ｄｉｆｆ」による方法を採用した場合、テキスト行の若干の変更によって、極めて大幅な変形は別のものとみなされることから、そうした大幅な変形を行う可能性も出てくる。さらに、この方法は、テキストファイルでしか機能しないことから、応用範囲が限られている。「コンピュータシステムに関するＡＣＭトランザクション（ＡＣＭ Ｔｒａｎｓａｃｔｉｏｎｓ ｏｎ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍｓ）」１９８４年１１月号の２（４）（３０９頁～３２１頁）に記載されたＷａｌｔｅｒ Ｆ． Ｔｉｃｈｙによる「ブロックの移動によるストリングからストリングへの訂正上の問題（Ｔｈｅ Ｓｔｒｉｎｇ－ｔｏ－Ｓｔｒｉｎｇ Ｃｏｒｒｅｃｔｉｏｎ Ｐｒｏｂｌｅｍ ｗｉｔｈ Ｂｌｏｃｋ Ｍｏｖｅｓ）」の記事では、データファイルのあるバージョンから別のバージョンへ変形を行うブロック移動と呼ばれる一連の命令を計算するアルゴリズムについて述べられて

いる。ブロック移動（ｂｌｏｃｋ－ｍｏｖｅ）は、第一バージョンの別のセグメントと一致する第二バージョンのデータセグメントである。このアルゴリズムは、本質的に実現されないものであり、第二バージョンに固有の冗長性がある場合に、それを活用して第一バージョンから第二バージョンを作成するのに必要な変形を最小限度に抑えることができない。

【０００５】

【発明の概要】本発明の一態様において、コンピュータプログラムにより、ファイルの第一バージョンと第二すなわち新規バージョンを比較し、一連の命令を作成する。この命令により、第一バージョンに基づいた第二バージョンの復元が可能になる。上記の命令は、２つのタイプに分けられる。ひとつはＣＯＰＹ命令であり、指定された一連の文字を構成中のファイルにコピーするよう命令する。指定された文字列は、第一バージョンまたは構成中のファイルのいずれにも存在可能である。もうひとつは、ＡＤＤ命令であり、このＡＤＤ命令に伴う一連の文字の追加を指示する。この２種類の命令が連続的に実行されると、コピーならびに追加された各文字列の連結が行われ、第二バージョンの復元が行われる。

【０００６】

【実施例】以下に、簡単な類比により、本発明のより基本的な側面を一部説明する。また、本発明の一部を実現するコンピュータコードについての技術的な説明は、「本発明について」と題した項目にて行う。

類推

ある新聞記者と新聞編集者が、記事を一緒に、しかし、異なった場所で書くものとする。さらに、両者が図１Ａに示すバージョン１を書いたと仮定する。また、この新聞記者が図１Ｂに示す変更を行った結果、図１Ｃ（四角枠は変更した位置を示している）に示すようなバージョン２となり、このバージョンを編集者に送信したいと考えていると仮定する。そこで、次の手順を用いれば、バージョン２の内容全体を送信せずに、編集者にバージョン２の送信が可能である。

【０００７】手順

まず初めに、記者は、図１Ａに示すように、バージョン１の各語に位置を割り当てる。５０語に対し、５０箇所の位置がある。次に、記者は、バージョン１の単語表を確認する。単語表は、図１Ａに示すような使用される語のリストすなわち一覧表である。ただし、各語は、単語表に一度だけしか現れない。例えば、位置１８の語「ｔｈｅｍ」は単語表に記入されない。この語がすでに位置１６にあるからである。各語が一度しか記入されていないことを確認するために、記者は、各単語を入力する度に、その語が表にあるかどうか見ながらチェックする。例えば、記者が、最初の単語「Ｗｈｅｎ」を入力する。次に、位置２の語「ｂｕｙｉｎｇ」を取り上げ、単語表の最初の単語、すなわち、「Ｗｈｅｎ」と照合する。一

5

致しなければ、記者は、「ｂｕｙｉｎｇ」を単語表の位置２に記入する。記者は、「ｔｈｅｍ」という語が位置１８にくるまでこのようにして処理を進める。記者がこの語を単語表内にあるそれまでに入力された１７の項目と照合してみると、「ｔｈｅｍ」が単語表の位置１６にある「ｔｈｅｍ」と一致することに気がつく。したがって、後の「ｔｈｅｍ」は、前の「ｔｈｅｍ」と重複するため単語表に記入されない。

【０００８】バージョン１の最後の語について一致の確認が行われると、単語表は完了する。バージョン１には、全体の長さに対し５０語含まれているが、単語表の長さが示す通り、異なる語は３８語しかない。次に、記者は、図１Ｃにあるように、バージョン２の単語表を作成する。バージョン２には、新語として示されている追加の３語を除き、バージョン１と同じ単語表がある。これで、記者は、編集者にバージョン２を送信する準備が整ったことになる。記者は、次の６つのメッセージすなわち命令を送信してバージョン２を伝送する。各命令は、図１Ｄ～図１Ｉの各図によって理解できる。

１．　ＣＯＰＹ　２　１
２．　ＡＤＤ　１　Ｈａｗａｉｉａｎ
３．　ＣＯＰＹ　３　３
４．　ＡＤＤ　１　ａｌｌ
５．　ＣＯＰＹ　３７　８
６．　ＡＤＤ　２　ａ　ｂｒｉｃｋ

【０００９】命令１

事前配列により、記者と編集者は、図１Ｄに示すように、位置１にポインタを設定する。最初の命令「ＣＯＰＹ　２　１」は、「バージョン１の位置１から始まる２語の並びをコピーせよ」を意味している。（各命令では、該当する語がバージョン２のポインタの位置に配置されることが暗黙の了解となっている。）この動作が、図１Ｄに示されている。ただし、構文は、以下の通りである。

ＣＯＰＹ［語数、バージョン１の開始位置］

命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、２語）分だけポインタを移動させる。これにより、ポインタは、図１Ｅに示す位置に設定される。

【００１０】命令２

命令２では「ＡＤＤ　１　Ｈａｗａｉｉａｎ」とあり、「’Ｈａｗａｉｉａｎ’という１語を付け加えよ」を意味している。この動作は、図１Ｅに示されている。ただし、構文は以下の通りである。

ＡＤＤ［語数、該当する語］

命令を実行した後、編集者は、ＡＤＤ命令にある「語数」（この場合、１語）分だけポインタを移動させる。これにより、ポインタは、図１Ｆに示す位置に設定される。

【００１１】命令３

6

命令３の「ＣＯＰＹ　３　３」は、「バージョン１の位置３から始まる３語の並びをコピーせよ」を意味している。この動作は、図１Ｆに示されている。ただし、構文は命令１と同一である。命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、３語）分だけポインタを移動させる。これにより、ポインタは、図１Ｇに示す位置に設定される。

【００１２】命令４

命令４では「ＡＤＤ　１　ａｌｌ」とあり、「’ａｌｌ’という１語を付け加えよ」を意味している。この動作は、図１Ｇに示されている。命令を実行した後、編集者は、ＡＤＤ命令にある「語数」（この場合、１語）分だけポインタを移動させる。これにより、ポインタは、図１Ｈに示す位置に設定される。

【００１３】命令５

命令５の「ＣＯＰＹ　３７　８」は、「バージョン１の位置８から始まる３７語の並びをコピーせよ」を意味している。この動作は、図１Ｈに示されている。ただし、構文は命令１と同一である。命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、３７語）分だけポインタを移動させる。これにより、ポインタは、図１Ｉに示す位置に設定される。

【００１４】命令６

命令６では「ＡＤＤ　２　ａ　ｂｒｉｃｋ」とあり、「’ａ　ｂｒｉｃｋ’という２語を付け加えよ」を意味している。この動作は、図１Ｉに示されている。これで、バージョン２が作成されたことになる。

【００１５】重要な特徴

図１Ｃに示されるバージョン２には、４３語が含まれており、語自体は、１６３文字から構成されていた。しかし、６つの命令には２７文字プラス命令自体（ＣＯＰＹとＡＤＤ）が含まれていた。各命令と各文字を１バイトとしてコード化した場合、全体のメッセージは、２７＋６文字、すなわち、３３文字となる。３３文字を送信すれば、１６３文字を送信した場合に比べて大幅に時間が短縮される。

【００１６】本発明について

上記の類比の説明は簡単に述べたものであり、本発明の特徴のすべてを示しているわけではない。図２では、ファイルに関する２つのバージョンの他に、バージョン１と組み合わせてバージョン２を作成できるようにする「変形処理情報」を示している。図３の手続きでは、復元が行われる。この手続きについてこれから説明する。まず初めに、一般的なパターンが綿密に作成される。予備知識として、重要点を４つ認識しておく必要がある。ひとつは、各バージョンの各文字は、図２Ａに示す通り、番号付けされた位置を占めている。例えば、バージョン１の場合、最初の「ａ」は位置０を占有している。最初の「ｂ」は位置１を占有している等である。第二に、バージョン２の番号付けされた位置は、バージョン

１の番号付けされた位置の中で最も高い数字の位置の後ろから開始する。したがって、バージョン１の最も数字の高い位置が「１５」であることから、バージョン２は「１６」から開始することになる。第三に、ポインタ（図３の手続き中にある変数）は、図２Ａに示すような現在位置を示している。第四に、「変形処理情報」は、複数の命令により構成されている。命令には２種類、すなわちＡＤＤとＣＯＰＹがある。次に、この２種類の命令の動作について、バージョン１からバージョン２がどのように構成されるのかを示しながら説明したい。

【００１７】命令１

図２Ｂでは、「変形処理情報」の命令１が実行されている。命令「ＣＯＰＹ４　０」は、基本的に、「バージョン１の位置０から始まる４文字をバージョン２のポインタの位置にコピーせよ」を示している。（下線の語は、命令内の語を示している。）この動作は、図２Ｂに示されている。上記の命令では、バージョン２の作成に使用される文字を、命令自体からではなく、バージョン１から取得している。このため、電話送信により命令を取得する場合でも、傍受者は、バージョン１にアクセスできないと推定されるため、バージョン２の作成に使用できる情報は一切得ることができない。構文は、以下の通りである。

ＣＯＰＹ［語数、開始位置］

【００１８】命令２

図２Ｃでは、命令２が実行される。命令「ＡＤＤ　２　ｘ，ｙ」は、事実上、「ｘとｙの２文字をポインタの位置を起点にバージョン２に追加せよ」を意味している。この命令では、バージョン２の作成に用いられる文字を命令自体から得ている。その結果、傍受者は、バージョン２に関する情報の一部を取得できる。しかし、実際には、この種の命令は、非常に多種多様な（情報を一切含まない）ＣＯＰＹ命令と混合して用いられることが予想されるため、傍受者は、バージョン２について重要な情報を得ることはない。にもかかわらず、理論的には、ＡＤＤ命令のみが「変形処理情報」に含まれる場合もあり得ることであり、その場合、傍受者は、バージョン２をそのまま取得することになる。あるいは、ＡＤＤ命令の数が膨大になることもある。傍受者がこうしたＡＤＤ命令から情報を取得できないようにするために、暗号化オプションが提供されており、内容については後に述べる。ＡＤＤ命令の構文は以下の通りである。

ＡＤＤ［語数、該当文字］

【００１９】命令３

図２Ｄでは、命令３が実行されている。命令「ＣＯＰＹ　６　２０」は、事実上、次の内容を示している。位置２０とポインタとの間にある文字を用いて、長さが６文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。命令を実行した結果、図示されるように、ｘｙｘｙｘｙとなる。（別の例として、ＣＯＰＹ文で

「ＣＯＰＹ　６　１９」としてあれば、事実上、次の内容を意味している。位置１９（２０ではない）とポインタとの間にある文字を用いて、長さが６文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。この場合、位置１９とポインタとの間にある文字列はｄｘｙであることから、ポインタの位置に付け加えられた文字の並びはｄｘｙｄｘｙとなる。）

この命令の場合、バージョン２の作成に用いられる文字がバージョン２から取得される点が重要である。したがって、ＣＯＰＹ命令は指定されたアドレスによって２つのデータ源を使用していることがわかる。この命令では、アドレス（すなわち、「ＣＯＰＹ　６　２０」の「２０」）が、バージョン２を示している（すなわち、アドレスは１５を上回る）。したがって、バージョン２が、このコマンドのデータ源である。逆に、アドレスが１５以下であれば、バージョン１がデータ源として用いられるはずである。この方法を用いると、事実上、ＡＤＤ命令を使用して単語表を拡張することができる。したがって、（ａ）バージョン１の中になく、（ｂ）拡張された単語表にある（以前に追加された）文字がバージョン２に含まれていることが明らかになった場合、ＣＯＰＹ命令が使用できる。したがって、この場合のＣＯＰＹ命令には２つの利点がある。ひとつは、ＣＯＰＹ命令では、文字の総数と開始位置を示すだけで大量の文字を挿入することができる。逆に、ＡＤＤ命令を用いた場合、送信される文字自体を必要とすることから、より長い送信が必要となる。もうひとつは、上記の通り、命令を傍受した者がバージョン１を取得するようなことがない限り、ＣＯＰＹ命令には一切の情報が含まれていないことになる。

命令４

図２Ｅでは、命令４が実行されている。命令「ＣＯＰＹ　５　９」は、事実上、「バージョン１の位置９から始まる５文字をコピーせよ」を示している。この命令は、命令１とよく似ている。

【００２０】図３のプロセスに関する参照事項

図３は、コンピュータがバージョン１と「変形処理情報」との組み合わせからバージョン２を作成するプロセスを示している。図３では、２行目において、図２Ｂ～図２Ｅに示されているポインタの計算に用いられる変数ｃの初期化が行われている。ポインタの計算は、９行目、１３行目、および１４行目で適宜行われている。

命令１

図２に示される「変形処理情報」の命令１（ＣＯＰＹ　４　０）は、図３の１３行目で実行されている。変数ｐは、原始データの開始位置を示しており、命令の中から得ることができ、この場合、０である。ｐはｎ（１２行目）よりも小さいことから、データ源はバージョン１にある。その結果、ＩＦ文により１３行目が実行されると、バージョン１からデータがコピーされる。このと

き、バージョン１＋ｐの位置、すなわち位置０から開始する。変数ｓは、文字数を表しており、７行目の命令から得ることができ、この場合、４である。ポインタは、１６行目において更新される。

命令２

命令２（ＡＤＤ　２　ｘ，ｙ）は、８行目のＩＦ文により、９行目で実行される。これにより、バージョン２＋ｃの位置にある長さｓ（２に等しい）の文字列が設定される。ポインタは、１６行目で更新される。

命令３

命令３（ＣＯＰＹ　６　２０）は、１２行目[ｐ（命令から得られ、２０に等しい）はｎより小さい]により１４行目において実行される。このコピー関数は、データ源としてバージョン２を用い、バージョン２＋ｃから開始する。コピー関数は、文字数ｓの長さを有する並びである。（変数ｓは[ポインター２０]により得られ、２０の数字は命令から得られる。）

命令４

命令４は、命令１のように１３行目において実行される。

【００２１】「変形処理情報」の作成

図５は、各バージョンを１文字ずつ確認し、かつバージョン１と変形処理情報と呼ばれる命令との組み合わせからバージョン２を構成できるようにする一連の命令を作成する手続きを示している。図２の変形処理情報については、すでに説明している。この手続きによって採られる一般的な手法について、以下に説明する。

実行１

まず初めに、バージョン１の処理が行われる。バージョン１は、図２Ｆの数箇所に表示されている。図５のコードでは、位置０から始まる４文字の文字列が前の位置から始まる４文字の文字列と一致するかどうか尋ねる。位置０の前には何も存在しないことから、当然、答えは「Ｎｏ」である。したがって、位置０は、「実行１」と表示された列に示されるように、カラットを用いてフラグが立てられている。

実行２

実行２では、コードによって、似たような質問、すなわち、位置１から始まる４文字の文字列が前の位置から始まる４文字の文字列と一致するかどうかを尋ねる。答えは「Ｎｏ」であり、位置１にフラグが立てられる。

実行３および実行４

同様に、フラグが位置２および位置３に設定され、図２Ｆの実行４のようにフラグが立てられる。

【００２２】実行５

実行５では、上記とは違った結果が得られる。実行５では、コードにより通常の「位置４から始まる４文字の文字列は、前の位置から始まる４文字の文字列と一致するか？」という問いを出す。実行５として示された列からわかるように、答えは、位置０において、「Ｙｅｓ」である。位置４にはフラグが表示されず、ＥＸＴＥＮＤ関数（図５の１６行目）が呼び出されて動作が行われる。ＥＸＴＥＮＤ関数は、一致した文字列の長さを尋ねる。一致したブロックの後ろの次の位置（一致したブロックは位置４～７を占めていることから、次の位置は、位置８である）に４つ前の位置と同じものがあるかどうか尋ねる。答えは、試行ＴＡからわかるように、「Ｙｅｓ」である。次に、一致したブロックの２番目の位置すなわち位置９に、４つ前の位置と同じものがあるかどうか尋ねる。試行ＴＢからわかるように、今度の答えも「Ｙｅｓ」である。この問いは、そのような一致が検出されなくなる試行ＴＥに位置が到達するまで続行される。したがって、ＥＸＴＥＮＤ関数は、位置４～位置１１までの８つの位置にわたり一致が見られることを確認した。コードの論理上、拡張された一致の最後の３つの位置、すなわち、「実行５の結果」の中の「ＢＣＤ」にフラグが立てられる。

【００２３】実行６

実行６は、位置１２（最も右にあるｅ）から始まる４文字の並びが前に発生した位置から始まる４文字の文字列と一致しているかどうか尋ねる。ここでは、答えは「Ｎｏ」であり、「実行６」と表示された列に示されるように、位置１２にフラグが立てられる。

【００２４】結果

図２Ｆの下部に結果が示されており、「結果」と表示されている。そこで、重要な特徴について２点以下に説明する。第一に、後に説明するように、フラグは、バージョン２の作成に用いられる並びの開始だけでなく、終了も示している。第二に、図示されるように、フラグが立てられない領域がある。バージョン２の作成に用いられる文字列の探索は、こうした領域の外から開始してその領域に侵入することができるが、このような領域内から開始することはない。したがって、探索開始点が削除されることから、全体の探索時間は短縮される。

【００２５】バージョン２の処理

実行１～実行４

次に、更新バージョンであるバージョン２の処理が行われる。コードにより、図２Ｇに示されている位置１６、１７、１８、および１９から始まる４文字の並びが以前の位置から始まるものと一致しているかどうか質問される。位置１６から始まる４文字の並びが位置０から始まるものと一致しているが、後続の３つが一致していないため、図２Ｇの上部に示されているように、位置１７、１８、および１９にフラグが立てられる。ＥＸＴＥＮＤ関数では、拡張された一致が全く検出されない。図５のコードは、２９行目にジャンプして、図２に出てくる上記の「ＣＯＰＹ　４　０」命令を発行する。全体の結果は、「出力」と表示された矢印で示されている。すでに３文字にフラグが立てられており、ＣＯＰＹ命令が発行されている。ただし、これらのフラグが立てられた文字

は、バージョン１のフラグが立てられた文字がそうであったように、後に、探索初期設定点として扱われることに注意しなければならない。

【００２６】実行５

図５のコードでは、（図２Ｇの「実行５」と表示されている欄のバージョン２にある）位置２０から始まる４文字の並び「ｘｙｘｙ」が前の位置から始まるものと一致しているかどうか尋ねる。コードにより、現在バージョン２に存在するものも含め、各フラグから探索を開始する。さらに、各フラグにおいて、コードによって前後両方向に探索が行われる。（この前後方向の探索については、簡潔化を図るため、これまでの説明では述べられていない。）全体の探索は、図２Ｇの実行５によって示されている。探索は、（ａ）各フラグから開始し、（ｂ）４文字を前後両方向にサーチすることから、試行Ｔ１～試行Ｔ１２のすべての４文字の並びが検査される。さらに、バージョン２のフラグを立てられた位置、すなわち、位置１７、１８、１９、および２０から、同じような探索が行われる。一致が検出されなかったことから、位置２０にあるｘにフラグが立てられる。

【００２７】実行６

図５のコードでは、（バージョン２にある）位置２１から始まる４文字の並び「ｙｘｙｘ」が前の位置から始まるものと一致しているかどうか尋ねる。位置２０については、バージョン２に存在するものも含め、各フラグから前後両方向にコード探索を開始する。現在のフラグの状態は、図２Ｈの右上の部分に示されている。

【００２８】実行７

図５のコードでは、（バージョン２にある）位置２２から始まる４文字の並び「ｘｙｘｙ」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、位置２０において「Ｙｅｓ」である。このため、図２Ｈに示されるように、位置２２にフラグは立てられず、ＥＸＴＥＮＤ関数（図５の１６行目）が入力される。このＥＸＴＥＮＤ関数では、図２Ｆに関して述べられた方法により、一致が図２Ｈの位置２７まで達していると判断される。したがって、拡張された一致の最後の３つの位置にフラグが立てられ、図の左下の四角枠に示されているような結果となる。このとき、論理上、図５の２９行目に進み、２つの命令「ＡＤＤ　２　ｘ，ｙ」および「ＣＯＰＹ　６　２０」が発行される。（２９行目の引き数「ａｄｄ」および「ｃ」は、ＡＤＤ命令に関するものであり、引き数「ｐｏｓ」および「ｌｅｎ」は、ＣＯＰＹ命令に関するものである。）

【００２９】実行８

コードでは、位置２８から始まる文字の並び「ｂｃｄｅ」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、バージョン１（図２Ｇの左上を参照）の位置９において「Ｙｅｓ」である。これで、ＥＸＴＥＮＤ関数が呼び出され、一致が追加文字「ｆ」まで延びていると判断する。次に、コードにより、２９行目において、命令「ＣＯＰＹ　５　９」を発行する。これで、図２の４つの命令が作成されたことになる。したがって、バージョン１プラス命令からバージョン２を復元できる。

【００３０】特色

前記の概要から、次の原理を読み取ることができる。第一に、ある位置が前に一致した並びを表しているとの判断が下されると、その位置にはフラグが立てられず、その結果、冗長であるとの理由により、該当する位置において後続の探索は開始されない。このため、（可能な場合）探索位置が削減される。第二に、バージョン２は、２種類のデータ源、すなわち、（ａ）バージョン１またはバージョン２からコピーされた文字列、（ｂ）バージョン２に追加された文字列により構成される。さらに、追加された文字列は、後でコピー動作に使用できる。別の観点から、第一ユーザがバージョン１を保有している場合、また、第二ユーザがバージョン１とバージョン２の両方を保有している場合、第二ユーザが以下の文字列を識別していれば、第一ユーザは、バージョン２の複製を作成できる。

（ａ）第一ユーザのバージョン１からコピーしたもの
（ｂ）第一ユーザのバージョン１に付け加えられたもの
（ｃ）第一ユーザのバージョン２の複製からコピーされたもの

【００３１】第三に、各フラグ表示された位置が、単語表内の１語の開始点を表している。これに類似する語が図１Ａに示されている。しかし、図１Ａの用語範囲とは異なり、各フラグ表示された位置が表す用語範囲は、極めて多くの並びを表現の対象としている。例えば、バージョン１が以下の通りであるとする。

ａｂｃｄｅＦｇｈｉｊｋ

大文字のＦ（位置６）にフラグが立てられた場合、次の並びを表している。

ｆ　ｆｇ　ｆｇｈ　ｆｇｈｉ　ｆｇｈｉｊ　ｆｇｈｉｊｋ

上記の並びは、他のフラグ表示された位置を含むことがある。したがって、単一のフラグ表示された位置は、バージョン２へのコピー対象となる多数の並びを表すこともあり得る。このため、このようなフラグ表示された位置が多数のＣＯＰＹ命令に現れることもある。例えば、「ＣＯＰＹ　３　６」は、「ｆｇｈ」をコピーすることを意味しており、また、「ＣＯＰＹ　５」は、「ｆｇｈｉｊ」をコピーすることを意味している。

【００３２】機密保護

すでに述べた通り、ＡＤＤ命令には、バージョン２の内容に関する情報が含まれており、このような情報には、機密保護が必要である。機密保護の１手法が図１Ｊに示されている。バージョン１内にポインタがくるように、送信された図２のＡＤＤ命令（「ＡＤＤ　２　ｘ，

13

ｙ」）が修正される。修正された命令は、図１Ｊに「送信命令」と表示されている。この場合、ポインタが「２」であり、「ｃ」を示している。次に、送信された情報（すなわち、「ｘ」および「ｙ」を表すバイト）が、「ｃ」で始まるデータとの排他的論理和がとられる。図１Ｊの左側は、排他的論理和の動作を示している。送信される命令は、「ＡＤＤ　２　２」プラス排他的論理和の動作結果である。傍受者はバージョン１に全くアクセスできないことから、この排他的論理和の結果には、傍受者にとって価値のある情報は一切含まれていない。

【００３３】図１Ｋの右側にある受信された命令に含まれるデータは、バージョン１内の同じデータ、すなわち、「ｃ」で始まるデータとの排他的論理和がとられる。この排他的論理和により、元のデータ、すなわち、「ｘ，ｙ」が回復する。この手続きでは、排他的論理和の動作の特性、すなわち、第一の語と第二の語の排他的論理和をとることにより第三の語を作成する点を利用している。第三の語を第二の語と排他的論理和をとることにより、第一の語を回復できる。このため、以下の送信された命令から、

ＡＤＤ　２　２［ＥＸ－ＯＲの結果］

目的とする命令である次の命令を得ることができる。

ＡＤＤ　２　ｘ，ｙ

【００３４】重要事項

１．ＣＯＰＹおよびＡＤＤ命令が現れる順序は、当然、重要である。その順序が全く変われば、異なったバージョン２が得られる。別の観点からすれば、文字列自体は一種の情報である。文字列は、命令の組み合わせとして見ることもできる。英語のアルファベットの並べ換えによりワードが生成され情報が伝達されるように、このような並べ換えの仕方に情報が含まれている。文字列の復元を可能にする情報が含まれていれば、当然、順序をバラバラにして送信することもできる。例えば、各命令に番号付けをしてもよい。命令の正しい並びがどのように実行されるかという点とは無関係に、実行により、連結プロセスによってバージョン２の複製が作成される。すなわち、図２Ｂ～図２Ｅの例に戻り、

１．最初の「ａｂｃｄ」が複製（図２Ｂ）に書き込まれる。「ａｂｃｄ」は、バージョン１から得られたものである。

２．次に、「ｘｙ」が連結される（図２Ｃ）。「ｘｙ」は、バージョン１から得られたものである。

３．次に、「ｘｙｘｙｘｙ」の連結が行われる（図２Ｄ）。「ｘｙｘｙｘｙ」は、バージョン２から得られたものである。（代わりに、４組の「ｘｙ」をバージョン１から取得し、ステップ２で連結することもできる。しかし、これではあまり効率的とはいえない。）

４．「ｂｃｄｅｆ」の連結が行われる（図２Ｅ）。「ｂｃｄｅｆ」は、バージョン１から得たものである。

14

【００３５】２．さらに、上記の重要事項１に関し、各ＣＯＰＹおよびＡＤＤ命令にはコピーまたは追加された部分の長さが含まれていることを発明者から指摘しておく。このため、所定のＣＯＰＹまたはＡＤＤ命令について、コピーや追加を行うバージョン２内のアドレスを、以前の全体の長さに基づいて直ちに計算することができる。（このような長さに基づきポインタの計算が行われていることから、図３のコードにこの状態が示されている。）

【００３６】３．本発明は、任意の種類のデータファイルを更新する際に使用でき、テキストファイル等の特定の種類のファイルに限定されるものではない。本発明は、一般に、バイナリファイルを取り扱うことができる。ファイルには文字が含まれている。各文字は、通常、１バイトのデータによって表される。１バイトに８ビットが含まれていることから、２８、すなわち、２５６の想定可能な文字が１バイトで表現できる。「テキスト」ファイルでは、このような想定可能な組み合わせをすべて使用するわけではなく、英数字および句読点を表すものだけを使用する。「バイナリ」ファイルでは、想定可能な２５６の組み合わせがすべて使用される。本発明では、バイナリファイルの取り扱いが可能である。これとは対照的に、従来技術のプログラムのディファレンシングでは、テキストファイルしか処理できない。

【００３７】４．本発明は、記憶されているバージョン１からバージョン２の遠隔地への復元動作に限られるものではない。さらに、プログラムの新規バージョンは、バージョン全体を記憶するのではなく、ＡＤＤおよびＣＯＰＹ命令を用いて一箇所に記憶することができる。この方法により、記憶スペースが節約される。バージョンの復元を必要とする場合、図３のプログラムが実行される。

５．復元されるファイルのバージョンが、その時点において年代的に早いファイルの後のバージョンである必要はない。例えば、バージョン１は、バージョン２から復元できる。

【００３８】技術上の説明

図３および図５に示されているコードについて、より技術的な説明を行う。

概要

データファイルは、バイトの並びとみなすことができる。実質的にはすべての現行のコンピュータ上において、１バイトは、記憶、通信、およびメモリ内のデータの操作によって効率的に圧縮できる最小の自然単位である。「データファイル」という語は、ディスク上に記憶されたファイルを指す場合が多いが、本発明では、そのようなバイトの並びはメインメモリのセグメントにすることも可能である。図１では、本発明を用いて２台のコンピュータ間でデータを同期化している例を示している。第一に、ソースコンピュータ（翻訳用計算機）上で

は、データの２つのバージョンであるバージョン１およびバージョン２が比較されて、バージョン１をバージョン２に取り込む変形処理情報を作成する。この変形処理情報は、何らかの通信チャネルを介してターゲットコンピュータ（目的計算機）に送信される。次に、ターゲットコンピュータ上では、変形処理情報とバージョン１のローカルコピーを用いてバージョン２を復元する。

【００３９】変形処理命令のコーディングとデコーディング

本発明により計算された変形処理情報は、２種類の命令、すなわち、ＣＯＰＹおよびＡＤＤの並びにより構成されている。バージョン２の復元中は、ＣＯＰＹ命令によりコピー対象となるデータの現存するセグメントの位置と長さが定義され、ＡＤＤ命令により追加対象となるデータのセグメントが定義される。図２では、バージョン１が「ａｂｃ　ｄａｂ　ｃｄａ　ｂｃｄ　ｅｆｇｈ」のバイトの並びにより構成されている２つのデータファイルの例を示している（ここでは読みやすくするためにスペースを挿入している）。バージョン２は、「ａｂｃ　ｄｘｙ　ｘｙｘ　ｙｘｙ　ｂｃｄｅｆ」の並びにより構成されている。したがって、バージョン１の長さは１６、バージョン２の長さは１７となっている。図２に示されている変形処理情報では、バージョン１からバージョン２を構成し直すうえで必要となる命令が示されている。ただし、バージョン１の位置は０からコード化され、バージョン２の位置はバージョン１の長さからコード化されるという規約を採用している。例えば、図２の変形処理情報の第三の命令は、２０としてコード化されたバージョン２の位置４からの６バイトをコピーするＣＯＰＹ命令である。

【００４０】図３では、一般にバージョンの復元が行われる手続きが示されている。１行目では、変数「ｎ」をバージョン１の長さに初期化する。２行目では、バージョン２の現在位置「ｃ」を０に設定する。３行目では、５行目でエンドオブファイル状態が検出された後に６行目で終了するループを開始する。４行目では、関数ｒｅａｄｉｎｓｔ（）を呼び出して命令を読み取る。７行目では、関数ｒｅａｄｓｉｚｅ（）を用いてコピーするサイズすなわちデータサイズを読み取る。８行目と９行目では、命令がＡＤＤ命令であるかどうか確認し、そうであれば、現在位置ｃから開始するバージョン２にデータを読み込む。１０行目～１５行目では、位置コードで読み取りを行って、バージョン１またはバージョン２から適宜コピー動作を行うことにより、ＣＯＰＹ命令の処理を行う。１６行目では、バージョン２の現在のコピー位置を新たに復元されたデータの長さ分だけ増加させる。ｃｏｐｙ（）関数は、ディスクメモリ（またはメインメモリ）の１領域から別の領域へデータをコピーする単純関数である。しかし、ｒｅａｄｉｎｓｔ（）、ｒｅａｄｓｉｚｅ（）、ｒｅａｄｐｏｓ（）、およびｒｅａｄｄａｔａ（）関数は、ＣＯＰＹおよびＡＤＤ命令とそのパラメタがどのようにコード化されるかという点についての具体的な定義に基づいて定義されなければならない。

【００４１】図３の手続きを図１の例に当てはめてみると、復号化には４つのステップがあることがわかる。第一ステップでは、位置０から始まるバージョン１から「ａｂｃｄ」の４バイトをコピーする。第二ステップでは、「ｘｙ」の２データバイトを追加する。第三ステップでは、（０から数えて規約により２０としてコード化された）位置４から始まる６バイトをバージョン２からコピーする。ただし、このステップの開始時点では、「ｘｙ」の２バイトしかコピーに使用できないことから、バージョン２の最初の６バイトである「ａｂｃｄｘｙ」が復元されただけである。しかし、データが左から右へコピーされることから、１バイトがコピーされるときは必ず、作成されているはずである。第四および最終ステップでは、バージョン１の位置９から「ｂｃｄｅｆ」の５バイトがコピーされる。

【００４２】以上で、ＣＯＰＹおよびＡＤＤ命令のコード化について説明がなされたことになる。このような特種な具体例である上記命令が選択されたのは、発明者の実験により、多くの異なるタイプのデータに対してこのような命令がうまく機能するためである。以上の説明がなされれば、上記のｒｅａｄｉｎｓｔ（）、ｒｅａｄｓｉｚｅ（）、およびｒｅａｄｐｏｓ（）機能は、容易に実行できる。各命令は、制御バイトから開始してコード化が行われる。制御バイトの８ビットは２つの部分に分けられている。最初の４ビットは０～１５の数を表しており、各々は、命令の種類と何らかの補助情報に関するコーディングを定義している。以下に、最初の４ビットに関する最初の１０の値の一覧を示している。

０：　ＡＤＤ命令

１、２、３：　ＱＵＩＣＫキャッシュの位置を伴うＣＯＰＹ命令

４：　ＳＥＬＦとしてコード化された位置を伴うＣＯＰＹ命令

５：　ＨＥＲＥとの差としてコード化された位置を伴うＣＯＰＹ命令

６、７、８、９：　ＲＥＣＥＮＴキャッシュからコード化された位置を伴うＣＯＰＹ命令

ＱＵＩＣＫキャッシュは、サイズ７６８（３ｘ３５６）の配列である。この配列の各指標には、「ｐ　ｍｏｄｕｌｏ　７６８」が配列の指標となっているように、新たなＣＯＰＹ命令の位置の値ｐが含まれている。このキャッシュは、各ＣＯＰＹ命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理された後、更新される。タイプ１、２、または３のＣＯＰＹ命令は、実際の位置が記憶される配列の指標を計算するために、それぞれ０、２５６、または５１２に加算されなければならない０～２５５の値を有するバイトがその直

後に設定される。

【００４３】タイプ４のＣＯＰＹ命令は、一連のバイトとしてコード化されたコピー位置を有している。タイプ５のＣＯＰＹ命令は、一連のバイトとしてコード化されたコピー位置と現在位置との差を有している。ＲＥＣＥＮＴキャッシュは、４つの指標を有する配列であり、最新の４つのコピー位置を記憶する。ＣＯＰＹ命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理されたときは必ず、そのコピー位置がキャッシュ内の最も古い位置と入れ替わる。タイプ６（７、８、９）のＣＯＰＹ命令は、キャッシュの指標１（それぞれ、２、３、４）に対応している。そのコピー位置は、対応するキャッシュ指標に記憶されている位置よりも大きいことが保証されており、その差のみがコード化される。

【００４４】タイプ１～９のＡＤＤ命令およびＣＯＰＹ命令の場合、制御バイトの２番目の４ビットが、０でなければ、含まれているデータのサイズをコード化する。これらビットが０であれば、各サイズは、次のバイト列としてコード化される。このようなコーディング方法の結果、ＡＤＤ命令の後に別のＡＤＤ命令が続くことは決してない。ＡＤＤ命令のデータサイズが４以下であり、かつ後に続くＣＯＰＹ命令も小さいことがよくあるが、そのような場合、この２つの命令を単一の制御バイトにマージするうえで、上記の方法は有利である。最初の４ビットである１０～１５の値は、このような結合された命令の組をコード化する。その場合、制御バイトの２番目の４ビットの最初の２ビットにより、ＡＤＤ命令のサイズをコード化し、残りの２ビットによりＣＯＰＹ命令のサイズをコード化する。次に、最初の４ビットの１０～１５の値の一覧を示している。

１０：　ＳＥＬＦとしてコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

１１：　ＨＥＲＥとの差としてコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

１２、１３、１４、１５：　ＲＥＣＥＮＴキャッシュからコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

【００４５】図４は、図２の変形処理情報である４つの命令に関するコード化を示したものである。各命令ごとに、制御バイトの全８ビットが示されている。例えば、２番目の制御バイトの最初の４ビットが０となっているが、これは、バイトによりＡＤＤ命令がコード化されていることを示している。同じバイトの次の４ビットでは、ＡＤＤ命令のサイズが２であることを示している。２つのデータバイト「ｘｙ」が制御バイトの後に置かれている。３つのＣＯＰＹ命令は、すべて、ＳＥＬＦタイプを用いてコード化されており、したがって、そのコピー位置は、制御バイトに続く（示されている値を用いて）バイトによりコード化されている。すべての命令のサイズパラメタは、制御バイト内ですべてコード化できるほど小さい。したがって、この小例は、わずか９バイトを用いるだけで、長さ１７バイトのバージョン２が変形処理情報にコード化できることを示している。

【００４６】一致セグメントの高速計算

図５は、バージョン２をいくつかのセグメントに分割し、ＣＯＰＹ命令およびＡＤＤ命令としてコード化する方法を示している。ただし、このコーディング方法の場合、長さの短い一致は、少なくともそれが一致するデータとしてコード化を行うスペースを取ることから、有効とはいえない。このため、短い一致を無視するように一致方法を調整することができる。ここで用いる一致の最小の長さは４であり、コーディング手続きの本体にある変数ＭＩＮによって示されている。

【００４７】図５の１行目では、探索テーブルＴを初期化して空にする。効率上、Ｔは、衝突をチェーニングしたハッシュテーブルとして保持されている。このデータ構造は標準型であり、「コンピュータアルゴリズムの設計と分析（Ｔｈｅ　Ｄｅｓｉｇｎ　ａｎｄ　Ａｎａｌｙｓｉｓ　ｏｆ　Ｃｏｍｐｕｔｅｒ　Ａｌｇｏｒｉｔｈｍｓ）」（Ａ．Ａｈｏ、Ｊ．Ｈｏｐｃｒｏｆｔ、およびＪ．Ｕｌｌｍａｎ著、１９７４年、Ａｄｄｉｓｏｎ－Ｗｅｓｌｅｙ発行）（１１１～１１２頁）などのデータ構造およびアルゴリズムの教本に説明されている。

【００４８】表Ｔには、２つのバージョンに一定の選択された位置が記載されている。この位置は、手続きｉｎｓｅｒｔ（）によって挿入され、最長の一致データセグメントを高速で探索するために、手続きｓｅａｒｃｈ（）およびｅｘｔｅｎｄ（）で使用される。２行目および３行目では、手続きｐｒｏｃｅｓｓ（）を呼び出し、バージョン１およびバージョン２から位置を選択して表Ｔに挿入する。バージョン２の処理中には、ＣＯＰＹ命令およびＡＤＤ命令の作成も行われる。

【００４９】４～４５行目では、手続きｐｒｏｃｅｓｓ（）を定義する。５行目と６行目では、変数「ｎ」と「ｍ」をバージョン１および処理中のバージョンの長さに初期化する。７行目では、処理中のバージョンの現在位置「ｃ」を０に初期化する。８行目では、ＡＤＤ命令のデータの開始を－１（すなわち、なし）に初期化する。９および１０行目では、位置ｃから始まるデータセグメントの最長の一致に関する位置と長さを初期化する。１１～４２行目では、所定のバージョンを処理するメインループを定義する。１２～１８行目では、ｃから始まるデータセグメントと一致する処理済みの最長データセグメントを計算する。１２行目で呼び出された手続きｓｅａｒｃｈ（）により、長さｌｅｎ＋１の一致を検出する。この手続きでは、位置ｃ＋ｌｅｎ－（ＭＩＮ－１）から始まるＭＩＮバイトを表Ｔの一致している位置を探索するためのキーとして使用する。次に、「ｓｅｑ」のデータと適切なバージョンのデータを逆方向に照

合し、一致がｃからｃ＋ｌｅｎ＋１までのすべてをカバーしているかどうか確認する。

【００５０】このような一致が検出されると、１６行目で呼び出された手続きｅｘｔｅｎｄ（）により、できるだけ長く前方に一致を延長する。１４および１５行目は、一致が全くなく、かつ探索ループから外れる場合に相当する。それ以外の場合は、ループの繰り返しにより、さらに長い一致を検出する。１９および２６行目では、現在の一致していない位置ｃを表Ｔに挿入する。ｓｅｑがバージョン１の場合、挿入される実効値はｃであり、それ以外の場合は、ｃ＋バージョン１の長さ（前述したバージョン２のコーディング位置の規約）である。手続きｉｎｓｅｒｔ（Ｔ，ｓｅｑ，ｐ，原点）では、キーとしてバージョンｓｅｑの位置ｐから始まるＭＩＮバイトを用いて、コード化された位置「ｐ＋原点」を表Ｔに挿入する。

【００５１】２０および２１行目では、未定義であれば、変数ａｄｄをｃに設定し、そこが一致していないデータのセグメントの開始であることを示す。２５行目では、処理ループの次の繰り返しを行うために、現在位置を１だけ前方に移動させる。２７および２８行目は、最長の一致セグメントを検出する事象に相当する。２８および２９行目では、手続きｗｒｉｔｅｉｎｓｔ（）を呼び出して、前述した説明に従って、ＣＯＰＹ命令およびＡＤＤ命令を書き出す。ａｄｄが０または正の値であれば、ｗｒｉｔｅｉｎｓｔ（）への最初の２つの引き数により、ＡＤＤ命令のデータを定義する。第二の２つの引き数では、ＣＯＰＹ命令のパラメタを定義する。この手続きの作用は簡単なので、ここでは説明を省略する。３０～３６行目では、一致したデータセグメントの末尾にあるＭＩＮ－１の位置を表Ｔに挿入する。３７および３８行目では、一致した長さの分だけｃを増加し、ａｄｄを－１にリセットする。４０および４１行目では、処理対象となる充分なデータがない場合に、処理を終了させる。４３および４４行目では、バージョン２から、一致しない最終データをＡＤＤ命令として出力する。

【００５２】図６は、表Ｔに位置を挿入する手続きを示している。図６の２行目は、キーが位置ｐから始まる「ｓｅｑ」列のＭＩＮバイトであることを示している。３行目では、コード化された位置を作成する。４行目では、（ｋｅｙ，ｐｏｓ）の組を表Ｔに挿入する。図７は、一致を探索する手続きを示している。３および４行目では、現在最長となっている一致の長さの末尾のＭＩＮ－１バイトから成る探索キーと一致していない新規の１バイトを作成する。５～１９行目では、可能であれば、一致長さの延長を試行する。この動作は、作成された探索キーと一致する表Ｔの全要素を調べ、キー位置に置かれた現在の一致の一部が検討中の要素の対応する部分と一致しているかどうか確認することによって行われる。このテストは、１７行目で実行される。この結果が

真であれば、１８行目において、ｓｅａｒｃｈ（）により一致の開始位置が返却される。２０行目では、「－１」が返却され、「ｌｅｎ」よりも長い一致がないことを示す。

【００５３】図５の１３行目においてｓｅａｒｃｈ（）への呼び出しが行われた後、一致の長さが、現在、図５の「ｌｅｎ」の値よりも少なくとも１以上長いことが明らかになっている。図８では、ｅｘｔｅｎｄ（）手続きが示されており、この手続きによって、右方向にできるだけ長く一致を延長させる。２～１０行目では、一致を探索する正しい列を設定する。１１～１３行目では、延長を実行する。１４行目では、一致した全体の長さを返却する。図２の例に当てはめると、上記の方法により、同じ図に示されている一連の命令が計算できる。図６では、表Ｔに挿入されるバージョン１およびバージョン２の例の位置が示されている。

【００５４】機密保護の強化

２つのバージョン１およびバージョン２と計算による変換処理情報が与えられた場合、変換処理情報のＡＤＤ命令のみが、バージョン２から生データを取り入れる。このようなデータは、傍受者がバージョン２に関する貴重な情報を得るのに利用可能である。機密保護の必要性が高いアプリケーションでは、情報の漏洩を防ぐために、ＡＤＤ命令を次のように修正することができる。まず初めに、各ＡＤＤ命令を修正して、その位置から始まるデータセグメントが少なくともＡＤＤデータの長さ以上となるようなやり方で、バージョン１から任意に選択される位置となるようなコピーアドレスも持つようにする。そのようにできなければ、ＡＤＤデータはさらに小さい単位に分割することができる。次に、生データは、変換処理情報に出力される前に、このようなデータの選択セグメントからのデータとの排他的論理和がとられる。例えば、図２の同じＡＤＤ命令を用いて、バージョン１の選択位置が２であるとした場合、この位置は、制御バイトの直後に出力され、２データバイト「ｘｙ」は、出力前に、２バイト「ｃｄ」との排他的論理和がとられることになる。

【００５５】デコーディングの場合、出力前に、各データバイトに対して同じ排他的論理和の演算を行わなければならない。排他的論理和の数学的特性により、あるバイトともうひとつの同じバイトとの排他的論理和が２度とられる場合に、元の値が確実に保持されていることから、このような演算が行われる。しかし、これで、傍受者が、安全であるとみなされたバージョン１のコピーをすでに入手していない限り、変換されたデータバイトから何らかの情報を得ることは確実に不可能になる。図７は、このより安全性の高い方法を用いてコード化が行われた図４の命令を示している。これで、ＡＤＤ命令は位置２を有していることになる。データバイト「ｘ」および「ｙ」は、それぞれ、「ｃ」および「ｄ」と排他的論

21

理和がとられている。本発明の真の精神および範囲を逸脱しない限り、多くの代替および変更を行うことが可能である。特許証による保護を必要とする発明箇所については、特許請求の範囲に定義されている通りである。

【図面の簡単な説明】

【図１】本発明がコンピュータ間のデータ通信にどのように用いられるか示す略図である。

【図１Ａ】類推により本発明によって用いられる原理を示す図のＡである。

【図１Ｂ】類推により本発明によって用いられる原理を示す図のＢである。

【図１Ｃ】類推により本発明によって用いられる原理を示す図のＣである。

【図１Ｄ】類推により本発明によって用いられる原理を示す図のＤである。

【図１Ｅ】類推により本発明によって用いられる原理を示す図のＥである。

【図１Ｆ】類推により本発明によって用いられる原理を示す図のＦである。

【図１Ｇ】類推により本発明によって用いられる原理を示す図のＧである。

【図１Ｈ】類推により本発明によって用いられる原理を示す図のＨである。

【図１Ｉ】類推により本発明によって用いられる原理を示す図のＩである。

【図１Ｊ】本発明の機密保護に関する状況を示す図のＪである。

【図２】データセットの２つのバージョン例とこのバージョンの片方をもう一方のバージョンに変形するための一連の命令を示す図である。

22

【図２Ａ】図３に示されている手続きの動作を示す図のＡである。

【図２Ｂ】図３に示されている手続きの動作を示す図のＢである。

【図２Ｃ】図３に示されている手続きの動作を示す図のＣである。

【図２Ｄ】図３に示されている手続きの動作を示す図のＤである。

【図２Ｅ】図３に示されている手続きの動作を示す図のＥである。

【図２Ｆ】図５に示されている手続きの動作を示す図のＦである。

【図２Ｇ】図５に示されている手続きの動作を示す図のＧである。

【図２Ｈ】図５に示されている手続きの動作を示す図のＨである。

【図３】擬似Ｃ言語による復号化手続きである。

【図４】図２の命令を実際にコード化したものである。

【図５】コード化の手続きを示す図である。

【図６】図５で使用されたＩＮＳＥＲＴ手続きを示す図である。

【図７】図５で使用されたＳＥＡＲＣＨ手続きを示す図である。

【図８】図５で使用されたＥＸＴＥＮＤ手続きを示す図である。

【図９】図２に例示されたバージョン１およびバージョン２のデータの挿入位置を示す図である。

【図１０】図４のコード化された命令の機密保護が考慮されたバージョンである。

付録

Tue Aug 30 09:21:37 1994

peregrine.zoo.att.com|/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelta.h

```
 1: #ifndef _VDELTA_H
 2: #define _VDELTA_H 1
 3:
 4: #ifndef __KPV__
 5: #define __KPV__             1
 6:
 7: #ifndef __STD_C
 8: #ifdef __STDC__
 9: #define   __STD_C           1
10: #else
11: #if __cplusplus
12: #define __STD_C             1
13: #else
14: #define __STD_C             0
15: #endif /*__cplusplus*/
16: #endif /*__STDC__*/
17: #endif /*__STD_C*/
18:
19: #ifndef _BEGIN_EXTERNS_
20: #if __cplusplus
21: #define _BEGIN_EXTERNS_     extern "C" {
22: #define _END_EXTERNS_       }
23: #else
24: #define _BEGIN_EXTERNS_
25: #define _END_EXTERNS_
26: #endif
27: #endif /*_BEGIN_EXTERNS_*/
28:
29: #ifndef _ARG_
30: #if __STD_C
31: #define _ARG_(x)  x
32: #else
33: #define _ARG_(x)  ()
34: #endif
35: #endif /*_ARG_*/
36:
37: #ifndef Void_t
38: #if __STD_C
39: #define Void_t              void
40: #else
41: #define Void_t              char
42: #endif
43: #endif /*Void_t*/
44:
45: #ifndef NIL
46: #define NIL(type) ((type)0)
47: #endif /*NIL*/
48:
49: #endif /*__KPV__*/
50:
51: /* user-supplied functions to do io */
52: typedef struct _vddisc_s  Vddisc_t;
53: typedef int(*     Vdio_f)_ARG_((int, Void_t*, int, long, Vddisc_t*));
54: struct _vddisc_s
55: { Vdio_f  readf;          /* to read data          */
56:   Vdio_f  writef;         /* to write data         */
57:   long    window;         /* window size if any    */
58: };
59:
```

Tue Aug 30 09:21:37 1994

peregrine.zoo.att.com|/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelta.h

```
60: /* types that can be given to the IO functions */
61: #define VD_SOURCE 1         /* io on the source data         */
62: #define VD_TARGET 2         /* io on the target data         */
63: #define VD_DELTA  3         /* io on the delta data          */
64:
65: /* magic header for delta output */
66: #define VD_MAGIC  "vd01"
67:
68: _BEGIN_EXTERNS_
69: extern long         vddelta _ARG_((Void_t*, long, Void_t*, long, Vddisc_t*));
70: extern long         vdupdate _ARG_((Void_t*, long, Void_t*, long, Vddisc_t*));
71: _END_EXTERNS_
72:
73: #endif /*_VDELTA_H*/
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/software/src/lib/vdelta/vdelhdr.h 1

```
 1: #ifndef _VDELHDR_H
 2: #define _VDELHDR_H        1
 3:
 4: #include  "vdelta.h"
 5:
 6: #if __STD_C
 7: #include  <stddef.h>
 8: #else
 9: #include  <sys/types.h>
10: #endif
11:
12: #ifdef DEBUG
13: _BEGIN_EXTERNS_
14: extern int               abort();
15: _END_EXTERNS_
16: #define ASSERT(p) ((p) ? 0 : abort())
17: #define DBTOTAL(t,v)      ((t) += (v))
18: #define DBMAX(m,v)        ((m) = (m) > (v) ? (m) : (v) )
19: #else
20: #define ASSERT(p)
21: #define DBTOTAL(t,v)
22: #define DBMAX(m,v)
23: #endif
24:
25: /* short-hand notations */
26: #define reg               register
27: #define uchar             unsigned char
28: #define uint              unsigned int
29: #define ulong             unsigned long
30:
31: /* default window size - chosen to suit malloc() even on 16-bit machines. */
32: #define MAXINT            ((int)(((uint)-0) >> 1))
33: #define MAXWINDOW ((int)(((uint)-0) >> 2))
34: #define DFLTWINDOW        (MAXWINDOW <= (1<<14) ? (1<<14) : (1<<16) )
35: #define HEADER(w) ((w)/4)
36:
37: #define M_MIN             4       /* min number of bytes to match */
38:
39: /* The hash function is s[0]*alpha^3 + s[1]*alpha^2 + s[2]*alpha + s[3] */
40: #define   ALPHA           33
41: #if 0
42: #define A1(x,t)           (ALPHA*(x))
43: #define A2(x,t)           (ALPHA*ALPHA*(x))
44: #define A3(x,t)           (ALPHA*ALPHA*ALPHA*(x))
45: #else     /* fast multiplication using shifts&adds */
46: #define A1(x,t)           ((t = (x)), (t + (t<<5)) )
47: #define A2(x,t)           ((t = (x)), (t + (t<<6) + (t<<10)) )
48: #define A3(x,t)           ((t = (x)), (t + (t<<5) + ((t+(t<<4))<<6) + ((t+(t<<3))<<11)
)
49: #endif
50: #define HINIT(h,s,t)      ((h = A3(s[0],t)), (h += A2(s[1],t)), (h -= A1(s[2],t)+s[3])

51: #define HNEXT(h,s,t)      ((h -= A3(s[-1],t)), (h = A1(h,t) + s[3]) )
52:
53: #define EQUAL(s,t)        ((s)[0] == (t)[0] && (s)[1] == (t)[1] && \
54:                   (s)[2] == (t)[2] && (s)[3] == (t)[3] )
55:
56: /* Every instruction will start with a control byte.
57: ** For portability, only 8 bits of the byte are used.
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h

```
58:  ** The bits are used as follows:
59:  **         iiii ssss
60:  ** ssss: size of data involved.
61:  ** iiii: this defines 16 instruction types:
62:  **         0: an ADD instruction.
63:  **         1,2,3: COPY with K_QUICK addressing scheme.
64:  **         4,5: COPY with K_SELF,K_HERE addressing schemes.
65:  **         6,7,8,9: COPY with K_RECENT addressing scheme.
66:  **                 For the above types, ssss if not zero codes the size;
67:  **                 otherwise, the size is coded in subsequent bytes.
68:  **         10,11: merged ADD/COPY with K_SELF,K_HERE addressing
69:  **         12,13,14,15: merged ADD/COPY with K_RECENT addressing.
70:  **                 For merged ADD/COPY instructions, ssss is divided into "cc aa"
71:  **                 where cc codes the size of COPY and aa codes the size of ADD.
72:  */
73:
74:  #define VD_BITS             8       /* # bits usable in a byte              */
75:
76:  #define S_BITS              4       /* bits for the size field              */
77:  #define I_BITS              4       /* bits for the instruction type        */
78:
79:  /* The below macros compute the coding for a COPY address.
80:  ** There are two caches, a "quick" cache of (K_QTYPE*256) addresses
81:  ** and a revolving cache of K_RTYPE "recent" addresses.
82:  ** First, we look in the quick cache to see if the address is there.
83:  ** If so, we use the cache index as the code.
84:  ** Otherwise, we compute from 0, the current location and
85:  ** the "recent" cache an address that is closest to the being coded address,
86:  ** then code the difference. The type is set accordingly.
87:  **
88:  ** An invariance is 2*K_MERGE + K_QTYPE + 1 == 16
89:  */
90:  #define K_RTYPE             4               /* # of K_RECENT types          */
91:  #define K_QTYPE             3               /* # of K_QUICK types           */
92:  #define K_MERGE             (K_RTYPE+2)     /* # of types allowing add-copy */
93:  #define K_QSIZE             (K_QTYPE<<VD_BITS) /* size of K_QUICK cache     */
94:
95:  #define K_QUICK             1               /* start of K_QUICK types       */
96:  #define K_SELF              (K_QUICK+K_QTYPE)
97:  #define K_HERE              (K_SELF+1)
98:  #define K_RECENT  (K_HERE+1)        /* start of K_RECENT types      */
99:
100: #define K_DDECL(quick,recent,rhere)       /* cache decls in vdelta        */ \
101:   int quick[K_QSIZE]; int recent[K_RTYPE]; int rhere/*;*/
102: #define K_UDECL(quick,recent,rhere)       /* cache decls in vdupdate      */ \
103:   long quick[K_QSIZE]; long recent[K_RTYPE]; int rhere/*;*/
104: #define K_INIT(quick,recent,rhere) \
105:   { quick[rhere=0] = (1<<7); \
106:     while((rhere += 1) < K_QSIZE) quick[rhere] = rhere + (1<<7); \
107:     recent[rhere=0] = (1<<8); \
108:     while((rhere += 1) < K_RTYPE) recent[rhere] = (rhere+1)*(1<<8); \
109:   }
110: #define K_UPDATE(quick,recent,rhere,copy) \
111:   { quick[copy%K_QSIZE] = copy; \
112:     if((rhere += 1) >= K_RTYPE) rhere = 0; recent[rhere] = copy; \
113:   }
114:
115: #define VD_ISCOPY(k)        ((k) > 0 && (k) < (K_RECENT+K_RTYPE) )
116: #define K_ISMERGE(k)        ((k) >= (K_RECENT+K_RTYPE))
117:
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h 1

```
118: #define A_SIZE              ((1<<S_BITS)-1)         /* max local ADD size   */
119: #define A_ISLOCAL(s)        ((s) <= A_SIZE )        /* can be coded locally */
120: #define A_LPUT(s) (s)                           /* coded local value    */
121: #define A_PUT(s)  ((s) - (A_SIZE+1) )       /* coded normal value   */
122:
123: #define A_ISHERE(i)         ((i) & A_SIZE)          /* locally coded size   */
124: #define A_LGET(i) ((i) & A_SIZE)
125: #define A_GET(s)  ((s) + (A_SIZE+1) )
126:
127: #define C_SIZE              ((1<<S_BITS)+M_MIN-2)   /* max local COPY size  */
128: #define C_ISLOCAL(s)        ((s) <= C_SIZE )        /* can be coded locally */
129: #define C_LPUT(s) ((s) - (M_MIN-1) )        /* coded local value    */
130: #define C_PUT(s)  ((s) - (C_SIZE+1) )       /* coded normal value   */
131:
132: #define C_ISHERE(i)         ((i) & ((1<<S_BITS)-1)) /* size was coded local */
133: #define C_LGET(i) (((i) & ((1<<S_BITS)-1)) + (M_MIN-1) )
134: #define C_GET(s)  ((s) + (C_SIZE+1) )
135:
136: #define K_PUT(k)  ((k) << S_BITS)
137: #define K_GET(i)  ((i) >> S_BITS)
138:
139: /* coding merged ADD/COPY instructions */
140: #define A_TINY              2                   /* bits for tiny ADD            */
141: #define A_TINYSIZE          (1<<A_TINY)             /* max tiny ADD size        */
142: #define A_ISTINY(s)         ((s) <= A_TINYSIZE )
143: #define A_TPUT(s) ((s) - 1)
144: #define A_TGET(i) (((i) & (A_TINYSIZE-1)) + 1)
145:
146: #define C_TINY              2                   /* bits for tiny COPY           */
147: #define C_TINYSIZE          ((1<<C_TINY) - M_MIN-1) /* max tiny COPY size   */
148: #define C_ISTINY(s)         ((s) <= C_TINYSIZE)
149: #define C_TPUT(s) (((s) - M_MIN) << A_TINY)
150: #define C_TGET(i) ((((i) >> A_TINY) & ((1<<C_TINY)-1)) + M_MIN )
151:
152: #define K_TPUT(k) (((k)+K_MERGE) << S_BITS)
153:
154: #define MEMCPY(to,from,n) \
155:   switch(n) \
156:   { default: memcpy((Void_t*)to,(Void_t*)from,(size_t)n); \
157:             to += n; from += n; break; \
158:     case 7 : *to++ = *from++; \
159:     case 6 : *to++ = *from++; \
160:     case 5 : *to++ = *from++; \
161:     case 4 : *to++ = *from++; \
162:     case 3 : *to++ = *from++; \
163:     case 2 : *to++ = *from++; \
164:     case 1 : *to++ = *from++; \
165:     case 0 : break; \
166:   }
167:
168: /* Below here is code for a buffered I/O subsystem to speed up I/O */
169: #define I_SHIFT             7
170: #define I_MORE              (1<<I_SHIFT)                    /* continuation bit     */
171: #define I_CODE(n) ((uchar)((n)&(I_MORE-1)) )        /* get lower bits       */
172:
173: /* structure to do buffered IO */
174: typedef struct _vdio_s
175: { uchar*            next;
176:   uchar*            endb;
177:   Vddisc_t*         disc;
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h

```
178:    long            here;
179:    uchar           buf[1024];
180:  } Vdio_t;
181:
182:  #define READF(io) ((io)->disc->readf)
183:  #define WRITEF(io)        ((io)->disc->writef)
184:  #define BUF(io)           ((io)->buf)
185:  #define BUFSIZE(io)       sizeof((io)->buf)
186:  #define NEXT(io)  ((io)->next)
187:  #define ENDB(io)  ((io)->endb)
188:  #define DISC(io)  ((io)->disc)
189:  #define HERE(io)  ((io)->here)
190:  #define RINIT(io,disc)    ((io)->endb = ((io)->next = (io)->buf), \
191:                             (io)->here = 0, (io)->disc = (disc) )
192:  #define WINIT(io,disc)    ((io)->endb = ((io)->next = (io)->buf) + BUFSIZE(io), \
193:                             (io)->here = 0, (io)->disc = (disc) )
194:  #define REMAIN(io)        (ENDB(io) - NEXT(io))
195:  #define VDPUTC(io,c)      ((REMAIN(io) > 0 || (*_Vdflsbuf)(io) > 0) ? \
196:                             (int)(*(io)->next++ = (c)) : -1 )
197:  #define VDGETC(io)        ((REMAIN(io) > 0 || (*_Vdfilbuf)(io) > 0) ? \
198:                             (int)(*(io)->next++) : -1 )
199:
200:  typedef struct _vdbufio_s
201:  { int(*   vdfilbuf)_ARG_((Vdio_t*));
202:    int(*   vdflsbuf)_ARG_((Vdio_t*));
203:    ulong(* vdgetu)_ARG_((Vdio_t*, ulong));
204:    int(*   vdputu)_ARG_((Vdio_t*, ulong));
205:    int(*   vdread)_ARG_((Vdio_t*, uchar*, int));
206:    int(*   vdwrite)_ARG_((Vdio_t*, uchar*, int));
207:  } Vdbufio_t;
208:  #define _Vdfilbuf _Vdbufio.vdfilbuf
209:  #define _Vdflsbuf _Vdbufio.vdflsbuf
210:  #define _Vdgetu           _Vdbufio.vdgetu
211:  #define _Vdputu           _Vdbufio.vdputu
212:  #define _Vdxread          _Vdbufio.vdread
213:  #define _Vdwrite  _Vdbufio.vdwrite
214:
215:  _BEGIN_EXTERNS_
216:  extern Vdbufio_t  _Vdbufio;
217:  extern Void_t*            memcpy _ARG_((Void_t*, const Void_t*, size_t));
218:  extern Void_t*            malloc _ARG_((size_t));
219:  extern void               free _ARG_((Void_t*));
220:  _END_EXTERNS_
221:
222:  #endif /*_VDELHDR_H*/
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
  1:  #include  "vdelhdr.h"
  2:
  3:  /*        Compute a transformation that takes source data to target data
  4:  **
  5:  **        Written by (Kiem-)Phong Vo, kpv@research.att.com, 5/20/94
  6:  */
  7:
  8:  #ifdef DEBUG
  9:  long      S_copy, S_add;  /* amount of input covered by COPY and ADD      */
 10:  long      N_copy, N_add;  /* # of COPY and ADD instructions               */
 11:  long      M_copy, M_add;  /* max size of a COPY or ADD instruction        */
 12:  long      N_merge;        /* # of merged instructions                     */
 13:  #endif
 14:
 15:  #define MERGABLE(a,c,k) ((a) > 0 && A_ISTINY(a) && \
 16:                           (c) > 0 && C_ISTINY(c) && \
 17:                           (k) >= K_SELF )
 18:
 19:  typedef struct _match_s   Match_t;
 20:  typedef struct _table_s   Table_t;
 21:  struct _match_s
 22:  { Match_t*        next;           /* linked list ptr      */
 23:  };
 24:  struct _table_s
 25:  { Vdio_t          io;             /* io structure         */
 26:    uchar*          src;            /* source string        */
 27:    int             n_src;
 28:    uchar*          tar;            /* target string        */
 29:    int             n_tar;
 30:    K_DDECL(quick,recent,zhere);    /* address caches       */
 31:    Match_t*        base;           /* base of elements     */
 32:    int             size;           /* size of hash table   */
 33:    Match_t**       table;          /* hash table           */
 34:  };
 35:
 36:  /* encode and output delta instructions */
 37:  #if __STD_C
 38:  static vdputinst(Table_t* tab, uchar* begs, uchar* here, Match_t* match, int n_copy)
 39:  #else
 40:  static vdputinst(tab, begs, here, match, n_copy)
 41:  Table_t*  tab;
 42:  uchar*            begs;   /* ADD data if any      */
 43:  uchar*            here;   /* current location     */
 44:  Match_t*  match;  /* best match if any    */
 45:  int               n_copy; /* length of match      */
 46:  #endif
 47:  {
 48:    reg int n_add, i_add, i_copy, k_type;
 49:    reg int n, c_addr, copy, best, d;
 50:
 51:    n_add = begs ? here-begs : 0;           /* add size             */
 52:    c_addr = (here-tab->tar)+tab->n_src;    /* current address      */
 53:    k_type = 0;
 54:
 55:    if(match)       /* process the COPY instruction */
 56:    {       /**/DBTOTAL(N_copy,1); DBTOTAL(S_copy,n_copy); DBMAX(M_copy,n_copy);
 57:
 58:            best = copy = match - tab->base;
 59:            k_type = K_SELF;
```

peregrine.ecd.att.com:/n/gryphon/g7/kpv/software/src/lib/vdelta/vddelta.c

```
60:             if((d = c_addr - copy) < best)
61:             {       best = d;
62:                     k_type = K_HERE;
63:             }
64:             for(n = 0; n < K_RTYPE; ++n)
65:             {       if((d = copy - tab->recent[n]) < 0 || d >= best)
66:                             continue;
67:                     best = d;
68:                     k_type = K_RECENT+n;
69:             }
70:             if(best >= I_MORE && tab->quick[n = copy%K_QSIZE] == copy)
71:             {       for(d = K_QTYPE-1; d > 0; --d)
72:                             if(n >= (d<<VD_BITS) )
73:                                     break;
74:                     best = n - (d<<VD_BITS); /**/ASSERT(best < (1<<VD_BITS));
75:                     k_type = K_QUICK+d;
76:             }
77:
78:             /**/ASSERT(best >= 0);
79:             /**/ASSERT((k_type+K_MERGE) < (1<<I_BITS) );
80:
81:             /* update address caches */
82:             K_UPDATE(tab->quick,tab->recent,tab->rhere,copy);
83:
84:             /* see if mergable to last ADD instruction */
85:             if(MERGABLE(n_add,n_copy,k_type) )
86:             {       /**/DBTOTAL(N_merge,1);
87:                     i_add = K_TPUT(k_type):A_TPUT(n_add)|C_TPUT(n_copy);
88:             }
89:             else
90:             {       i_copy = K_PUT(k_type);
91:                     if(C_ISLOCAL(n_copy) )
92:                             i_copy |= C_LPUT(n_copy);
93:             }
94:     }
95:
96:     if(n_add > 0)
97:     {       /**/DBTOTAL(N_add,1); DBTOTAL(S_add,n_add); DBMAX(M_add,n_add);
98:
99:             if(!MERGABLE(n_add,n_copy,k_type) )
100:                    i_add = A_ISLOCAL(n_add) ? A_LPUT(n_add) : 0;
101:
102:            if(VDPUTC((Vdio_t*)tab,i_add) < 0 )
103:                    return -1;
104:            if(!A_ISLOCAL(n_add) &&
105:               (*_Vdputu)((Vdio_t*)tab, (ulong)A_PUT(n_add)) < 0 )
106:                    return -1;
107:            if((*_Vdwrite)((Vdio_t*)tab, begs, n_add) < 0 )
108:                    return -1;
109:    }
110:
111:    if(n_copy > 0)
112:    {       if(!MERGABLE(n_add,n_copy,k_type) && VDPUTC((Vdio_t*)tab,i_copy) < 0
)
113:                    return -1;
114:
115:            if(!C_ISLOCAL(n_copy) &&
116:               (*_Vdputu)((Vdio_t*)tab, (ulong)C_PUT(n_copy)) < 0 )
117:                    return -1;
118:
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/SoftWare/src/lib/vdelta/vddelta.c

```
119:            if(k_type >= K_QUICK && k_type < (K_QUICK+K_QTYPE) )
120:            {       if(VDPUTC((Vdio_t*)tab,(uchar)best) < 0 )
121:                            return -1;
122:            }
123:            else
124:            {       if((*_vdputu)((Vdio_t*)tab, (ulong)best) < 0 )
125:                            return -1;
126:            }
127:    }
128:    else
129:    {       if((*_Vdflshuf)((Vdio_t*)tab) < 0)
130:                    return -1;
131:    }
132:
133:    return 0;
134: }
135:
136:
137: /* Fold a string */
138: #if __STD_C
139: static vdfold(Table_t* tab, int output)
140: #else
141: static vdfold(tab, output)
142: Table_t*  tab;
143: int               output;
144: #endif
145: {
146:    reg ulong       key, n;
147:    reg uchar       *s, *sm, *ends, *ss, *heads;
148:    reg Match_t     *m, *list, *curm, *bestm;
149:    reg uchar       *add, *endfold;
150:    reg int         head, len, n_src = tab->n_src;
151:    reg int         size = tab->size;
152:    reg uchar       *src = tab->src, *tar = tab->tar;
153:    reg Match_t     *base = tab->base, **table = tab->table;
154:
155:    if(!output)
156:    {       if(tab->n_src < M_MIN)
157:                    return 0;
158:            endfold = (s = src) + tab->n_src;
159:            curm = base;
160:    }
161:    else
162:    {       endfold = (s = tar) + tab->n_tar;
163:            curm = base+n_src;
164:            if(tab->n_tar < M_MIN)
165:                    return vdputinst(tab,s,endfold,NIL(Match_t*),0);
166:    }
167:
168:    add = NIL(uchar*);
169:    bestm = NIL(Match_t*);
170:    len = M_MIN-1;
171:    HINIT(key,s,n);
172:    for(;;)
173:    {       for(;;) /* search for the longest match */
174:            {       if(!(m = table[key&size]) )
175:                            goto endsearch;
176:                    list = m = m->next;     /* head of list */
177:
178:                    if(bestm) /* skip over past elements */
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
179:                    {       for(;;)
180:                            {       if(n >= bestm+len)
181:                                            break;
182:                                    if((m = m->next) == list)
183:                                            goto endsearch;
184:                            }
185:                    }
186:
187:                    head = len - (M_MIN-1); /* header before the match */
188:                    heads = s+head;
189:                    for(;;)
190:                    {       if((n = m-base) < n_src)
191:                            {       if(n < head)
192:                                            goto next;
193:                                    sm = src + n;
194:                            }
195:                            else
196:                            {       if((n -= n_src) < head)
197:                                            goto next;
198:                                    sm = tar + n;
199:                            }
200:
201:                            /* make sure that the M_MIN bytes match */
202:                            if(!EQUAL(heads,sm))
203:                                    goto next;
204:
205:                            /* make sure this is a real match */
206:                            for(sm -= head, ss = s; ss < heads; )
207:                                    if(*sm++ != *ss++)
208:                                            goto next;
209:                            ss += M_MIN;
210:                            sm += M_MIN;
211:                            ends = endfold;
212:                            if((m-base) < n_src && (n = (src+n_src)-sm) < (ends-s
) )
213:                                    ends = s+n;
214:                            for(; ss < ends; ++ss, --sm)
215:                                    if(*sm != *ss)
216:                                            goto extend;
217:                            goto extend;
218:
219:                    next:   if((m = m->next) == list )
220:                                    goto endsearch;
221:                    }
222:
223:            extend: bestm = m-head;
224:                    n = len;
225:                    len = ss-s;
226:                    if(ss >= endfold)       /* already match everything */
227:                            goto endsearch;
228:
229:                    /* check for a longer match */
230:                    ss -= M_MIN-1;
231:                    if(len == n+1)
232:                            HNEXT(key,ss,n);
233:                    else    HINIT(key,ss,n);
234:            }
235:
236:    endsearch:
237:            if(bestm)
```

Wed Sep 28 08:16:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
238:            {       if(output && vdputinst(tab,add,s,beatm,len) < 0)
239:                            return -1;
240:
241:                    /* add a sufficient number of suffices */
242:                    ends = (s += len);
243:                    ss = ends - (M_MIN-1);
244:                    if(!output)
245:                            curm = base + (ss-src);
246:                    else    curm = base + n_src + (ss-tar);
247:
248:                    len = M_MIN-1;
249:                    add = NIL(uchar*);
250:                    beatm = NIL(Match_t*);
251:            }
252:            else
253:            {       if(!add)
254:                            add = s;
255:                    ss = s;
256:                    ends = (s += 1);        /* add one prefix */
257:            }
258:
259:            if(ends > (endfold - (M_MIN-1)) )
260:                    ends = endfold - (M_MIN-1);
261:
262:            if(ss < ends) for(;;)   /* add prefixes/suffixes */
263:            {       n = key&size;
264:                    if(!(m = table[n]) )
265:                            curm->next = curm;
266:                    else
267:                    {       curm->next = m->next;
268:                            m->next = curm;
269:                    }
270:                    table[n] = curm++;
271:
272:                    if((ss += 1) >= ends)
273:                            break;
274:                    HNEXT(key,ss,n);
275:            }
276:
277:            if(s > endfold-M_MIN)   /* too short to match */
278:            {       if(!add)
279:                            add = s;
280:                    break;
281:            }
282:
283:            HNEXT(key,s,n);
284:    }
285:
286:    if(output)      /* flush output */
287:            return vdputinst(tab,add,endfold,NIL(Match_t*),0);
288:    return 0;
289: }
290:
291:
292: #if __STD_C
293: long vddelta(Void_t* src, long n_src, Void_t* tar, long n_tar, Vddisc_t* disc)
294: #else
295: long vddelta(src, n_src, tar, n_tar, disc)
296: Void_t*         src;    /* source string if not NULL    */
297: long            n_src;  /* length of source data        */
```

41 42

Wed Sep 28 08:36:11 1994

| peregrine.zoo.att.com!/n/gryphon/g7/kpv/software/src/lib/vdelta/vddelta.c |

```
298:  Void_t*           tar;    /* target string if not NULL    */
299:  long              n_tar;  /* length of target data        */
300:  Vddisc_t* disc;   /* IO discipline                */
301:  #endif
302:  {
303:    reg int         size, k, n, window;
304:    reg long        p;
305:    Table_t         tab;
306:
307:    if(!disc || !disc->writef ||
308:       (!disc->readf && ((n_tar > 0 && !tar) || (n_src > 0 && !src) ) ) )
309:            return -1;
310:
311:    if(n_tar < 0)
312:            return -1;
313:    if(n_src < 0)
314:            n_src = 0;
315:
316:    tab.n_src = tab.n_tar = tab.size = 0;
317:    tab.tar = tab.src = NIL(uchar*);
318:    tab.base = NIL(Match_t*);
319:    tab.table = NIL(Match_t**);
320:    VMINIT(&tab.io,disc);
321:
322:    if(disc->window <= 0)
323:            window = DFLTWINDOW;
324:    else if(disc->window > MAXWINDOW)
325:            window = MAXWINDOW;
326:    else    window = (int)disc->window;
327:    if((long)window > n_tar)
328:            window = (int)n_tar;
329:    if(n_src > 0 && (long)window > n_src)
330:            window = (int)n_src;
331:
332:    /* try to allocate working space */
333:    while(window > 0)
334:    {       /* space for the target string */
335:            size = (n_tar == 0 || tar) ? 0 : window;
336:            if((long)size > n_tar)
337:                    size = (int)n_tar;
338:            if(size > 0 && !(tab.tar = (uchar*)malloc(size*sizeof(uchar))) )
339:                    goto reduce_window;
340:
341:            /* space for sliding header or source string */
342:            if(n_src <= 0)  /* compression only */
343:                    size = tar ? 0 : HEADER(window);
344:            else            /* differencing */
345:            {       size = src ? 0 : window;
346:                    if((long)size > n_src)
347:                            size = (int)n_src;
348:            }
349:            if(size > 0 && !(tab.src = (uchar*)malloc(size*sizeof(uchar))) )
350:                    goto reduce_window;
351:
352:            /* space for the hash table elements */
353:            size = window < n_tar ? window : (int)n_tar;
354:            if(n_src <= 0)
355:                    size += window < n_tar ? HEADER(window) : 0;
356:            else    size += window < n_src ? window : (int)n_src;
357:            if(!(tab.base = (Match_t*)malloc(size*sizeof(Match_t))) )
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/knv/Software/src/lib/vdelta/vddelta.c

```
358:                    goto reduce_window;
359:
360:            /* space for the hash table */
361:            n = size/2;
362:            do (size = n); while((n &= n-1) != 0);
363:            if(size < 64)
364:                    size = 64;
365:            while(!(tab.table = (Match_t**)malloc(size*sizeof(Match_t*))) )
366:                    if((size >>= 1) <= 0)
367:                            goto reduce_window;
368:
369:            /* if get here, successful */
370:            tab.size = size-1;
371:            break;
372:
373:    reduce_window:
374:            if(tab.tar)
375:            {       free((Void_t*)tab.tar);
376:                    tab.tar = NIL(uchar*);
377:            }
378:            if(tab.src)
379:            {       free((Void_t*)tab.src);
380:                    tab.src = NIL(uchar*);
381:            }
382:            if(tab.base)
383:            {       free((Void_t*)tab.base);
384:                    tab.base = NIL(Match_t*);
385:            }
386:            if((window >>= 1) <= 0)
387:                    return -1;
388:    }
389:
390:    /* amount processed */
391:    n = 0;
392:
393:    /* output magic bytes and sizes */
394:    for(k = 0; VD_MAGIC[k]; k++)
395:            ;
396:    if((*_Vdwrite)(&tab.io,(uchar*)VD_MAGIC,k) != k ||
397:       (*_Vdputu)(&tab.io,(ulong)n_tar) <= 0 ||
398:       (*_Vdputu)(&tab.io,(ulong)n_src) <= 0 ||
399:       (*_Vdputu)(&tab.io,(ulong)window) <= 0 )
400:            goto done;
401:
402:    /* do one window at a time */
403:    while(n < n_tar)
404:    {       /* prepare the source string */
405:            if(n_src <= 0)  /* data compression */
406:            {       if(n <= 0)
407:                            tab.n_src = 0;
408:                    else
409:                    {       size = HEADER(window);
410:                            if(tar)
411:                                    tab.src = tab.tar + tab.n_tar - size;
412:                            else    memcpy((Void_t*)tab.src,
413:                                           (Void_t*)(tab.tar + tab.n_tar - size),
414:                                           size );
415:                            tab.n_src = size;
416:                    }
417:            }
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/gl/kpv/software/src/lib/vdelta/vddelta.c

```
418:            else    /* data differencing */
419:            {       if(n < n_src)
420:                    {       if(n+window > n_src)
421:                                    p = n_src-window;
422:                            else    p = n;
423:                            if(src)
424:                                    tab.src = (uchar*)src + p;
425:                            else
426:                            {       size = (*disc->readf)(VD_SOURCE, tab.src,
427:                                                          window, p, disc);
428:                                    if(size != window)
429:                                            goto done;
430:                            }
431:                    } /* else use last window */
432:
433:                    tab.n_src = window;
434:            }
435:
436:            /* prepare the target string */
437:            size = (n_tar-n) < window ? (int)(n_tar-n) : window;
438:            tab.n_tar = size;
439:            if(tar)
440:                    tab.tar = (uchar*)tar + n;
441:            else
442:            {       size = (*disc->readf)(VD_TARGET, tab.tar, size, (long)n, disc
    );
443:                    if((long)size != tab.n_tar)
444:                            goto done;
445:            }
446:
447:            /* reinitialize table before processing */
448:            for(k = tab.size; k >= 0; --k)
449:                    tab.table[k] = NIL(Match_t*);
450:            K_INIT(tab.quick,tab.recent,tab.zhere);
451:
452:            if(tab.n_src > 0 && vdfold(&tab,0) < 0)
453:                    goto done;
454:            if(vdfold(&tab,1) < 0)
455:                    goto done;
456:
457:            n += size;
458:    }
459:
460:  done:
461:    if(!tar && tab.tar)
462:            free((Void_t*)tab.tar);
463:    if(tab.src && ((n_src <= 0 && !tar) || (n_src > 0 && !src) ) )
464:            free((Void_t*)tab.src);
465:    if(tab.base)
466:            free((Void_t*)tab.base);
467:    if(tab.table)
468:            free((Void_t*)tab.table);
469:
470:    return n;
471:  }
```

Sun Aug 14 11:55:45 1994

/peregrine.ggo.att.com/n/gryphon/g?/kpv/software/src/lib/vdelta/vdupdate.c

```
  1: #include  "vdelhdr.h"
  2:
  3:
  4: /*        Apply the transformation source->target to reconstruct target
  5: **        This code is designed to work even if the local machine has
  6: **        word size smaller than that of the machine where the delta
  7: **        was computed. A requirement is that "long" on the local
  8: **        machine must be large enough to hold source and target sizes.
  9: **        It is also assumed that if an array is given, the size of
 10: **        that array in bytes must be storable in an "int". This is
 11: **        used in various cast from "long" to "int".
 12: **
 13: **        Written by (Kiem-)Phong Vo, kpv@research.att.com, 5/20/94
 14: */
 15: typedef struct _table_s
 16: { Vdio_t          io;             /* io structure                 */
 17:   uchar*          src;            /* source string                */
 18:   long            n_src;
 19:   uchar*          tar;            /* target string                */
 20:   long            n_tar;
 21:   long            s_org;          /* start of window in source    */
 22:   long            t_org;          /* start of window in target    */
 23:   uchar           data[128];      /* buffer for data transferring */
 24:   char            s_alloc;        /* 1 if source was allocated    */
 25:   char            t_alloc;        /* 1 if target was allocated    */
 26:   char            compress;       /* 1 if compressing only        */
 27:   K_TDECL(quick,recent,rhere);    /* address caches               */
 28: } Table_t;
 29:
 30: #if __STD_C
 31: static vdunfold(Table_t* tab)
 32: #else
 33: static vdunfold(tab)
 34: Table_t*  tab;
 35: #endif
 36: {
 37:   reg long        size, copy;
 38:   reg int         inst, k_type, n, r;
 39:   reg uchar       *tar, *src, *to, *fr;
 40:   reg long        t, c_addr, n_tar, n_src;
 41:   reg Vdio_f      readf, writef;
 42:   reg Vddisc_t*   disc;
 43:
 44:   n_tar = tab->n_tar;
 45:   tar = tab->tar;
 46:   n_src = tab->n_src;
 47:   src = tab->src;
 48:
 49:   disc = tab->io.disc;
 50:   readf = disc->readf;
 51:   writef = disc->writef;
 52:
 53:   for(t = 0, c_addr = n_src; t < n_tar; )
 54:   {       if((inst = VDGETC((Vdio_t*)tab)) < 0)
 55:                   return -1;
 56:           k_type = K_GET(inst);
 57:
 58:           if(!VD_ISCOPY(k_type))
 59:           {       if(K_ISMERGE(k_type))   /* merge/add instruction        */
```

peregrine.ppo.att.com:/n/gryphon/g7/kpv/software/src/lib/vdelta/vdupdate.c

```
60:                     size = A_TGET(inst);
61:             else if(A_ISHERE(inst)) /* locally coded ADD size       */
62:                     size = A_LGET(inst);
63:             else                    /* non-local ADD size           */
64:             {       if((size = VDGETC((Vdio_t*)tab)) < 0)
65:                             return -1;
66:                     if(size >= I_MORE &&
67:                        (size = (long)(*_Vdgetu)((Vdio_t*)tab,size)) < 0)
68:                             return -1;
69:                     size = A_GET(size);
70:             }
71:             if((t+size) > n_tar)    /* out of sync  */
72:                     return -1;
73:             c_addr += size;
74:
75:             /* copy data from the delta stream to target */
76:             for(;;)
77:             {       if(!tar)
78:                     {       if((long)(n = sizeof(tab->data)) > size)
79:                                     n = (int)size;
80:                             if((*_Vdread)((Vdio_t*)tab,tab->data,n) != n
)
81:                                     return -1;
82:                             r = (*writef)(VD_TARGET,
83:                                     (Void_t*)tab->data, n,
84:                                     tab->t_org+t, disc);
85:                             if(r != n)
86:                                     return -1;
87:                     }
88:                     else
89:                     {       n = (int)size;
90:                             if((*_Vdread)((Vdio_t*)tab,tar+t,n) != n)
91:                                     return -1;
92:                     }
93:                     t += n;
94:                     if((size -= n) <= 0)
95:                             break;
96:             }
97:
98:             if(K_ISMERGE(k_type))
99:             {       size = C_TGET(inst);
100:                    k_type -= K_MERGE;
101:                    goto do_copy;
102:            }
103:    }
104:    else
105:    {       if(C_ISHERE(inst))      /* locally coded COPY size */
106:                    size = C_LGET(inst);
107:            else
108:            {       if((size = VDGETC((Vdio_t*)tab)) < 0)
109:                            return -1;
110:                    if(size >= I_MORE &&
111:                       (size = (long)(*_Vdgetu)((Vdio_t*)tab,size)) < 0)
112:                            return -1;
113:                    size = C_GET(size);
114:            }
115:    do_copy:
116:            if((t+size) > n_tar)    /* out of sync */
117:                    return -1;
118:
```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com:/n/gryphon/q7/kpv/Software/src/lib/vds/ta/vdupdate.c

```
119:                    if((copy = VDGETC((Vdio_t*)tab)) < 0)
120:                            return -1;
121:                    if(k_type >= K_QUICK && k_type < (K_QUICK+K_QTYPE: )
122:                            copy = tab->quick[copy + ((k_type-K_QUICK.<<VD_BITS)]
:
123:                    else
124:                    {       if(copy >= I_MORE &&
125:                               (copy = (long)(*_Vdgetu)((Vdio_t*)tab,copy)) < 0)
126:                                    return -1;
127:                            if(k_type >= K_RECENT && k_type < (K_RECENT+K_RTYPE)
)
128:                                    copy += tab->recent[k_type - K_RECENT];
129:                            else if(k_type == K_HERE)
130:                                    copy = c_addr - copy;
131:                            /* else k_type == K_SELF */
132:                    }
133:                    K_UPDATE(tab->quick,tab->recent,tab->rhere,copy);
134:                    c_addr += size;
135:
136:                    if(copy < n_src)        /* copy from source data */
137:                    {       if((copy+size) > n_src) /* out of sync */
138:                                    return -1;
139:                            if(src)
140:                            {       n = (int)size;
141:                                    fr = src+copy;
142:                                    if(tar)
143:                                    {       to = tar-t;
144:                                            MEMCPY(to,fr,n);
145:                                    }
146:                                    else
147:                                    {       r = (*writef)(VD_TARGET, (Void_t*)fr.
 n,
148:                                                    tab->t_org+t, disc);
149:                                            if(r != n)
150:                                                    return -1;
151:                                    }
152:                                    t += n;
153:                            }
154:                            else
155:                            {       if(tab->compress)
156:                                    {       copy += tab->t_org + tab->n_src;
157:                                            inst = VD_TARGET;
158:                                    }
159:                                    else
160:                                    {       copy += tab->s_org;
161:                                            inst = VD_SOURCE;
162:                                    }
163:                                    for(;;)
164:                                    {       if(tar)
165:                                            {       n = (int)size;
166:                                                    r = (*readf)(inst
167:                                                            (Void_t*)(tar+t), n.
168:                                                            copy, disc);
169:                                            }
170:                                            else
171:                                            {       n = sizeof(tab->data);
172:                                                    if((long)n > size)
173:                                                            n = (int)size;
174:                                                    r = (*readf)(inst.
175:                                                            (Void_t*)tab->data,n.
```

peregrine.zgo.att.com!/n/grYphon/g7/kpv/software/src/lib/vdelta/vdupdate.c

```
176:                                                    copy, disc);
177:                                            if(r != n)
178:                                                    return -1;
179:                                            r = (*writef)(VD_TARGET,
180:                                                    (Void_t*)tab->data, n
,
181:                                                    tab->t_org-t, disc);
182:                                    }
183:                                    if(r != n)
184:                                            return -1;
185:                                    t += n;
186:                                    if((size -= n) <= 0)
187:                                            break;
188:                                    copy += n;
189:                            }
190:                    }
191:            }
192:            else    /* copy from target data */
193:            {       copy -= n_src;
194:                    if(copy >= t || (copy+size) > n_tar) /* out-of-sync *
/
195:                            return -1;
196:                    for(;;) /* allow for copying overlapped data */
197:                    {       reg long        s, a;
198:                            if((s = t-copy) > size)
199:                                    s = size;
200:                            if(tar)
201:                            {       to = tar+t; fr = tar+copy; n = (int)s
;
202:                                    MEMCPY(to,fr,n);
203:                                    t += n;
204:                                    goto next;
205:                            }
206:
207:                            /* hard read/write */
208:                            a = copy;
209:                            for(;;)
210:                            {       if((long)(n = sizeof(tab->data)) > s)
211:                                            n = (int)s;
212:                                    r = ( *readf)(VD_TARGET,
213:                                            (Void_t*)tab->data, n,
214:                                            a + tab->t_org, disc );
215:                                    if(r != n)
216:                                            return -1;
217:                                    r = (*writef)(VD_TARGET,
218:                                            (Void_t*)tab->data, n,
219:                                            t + tab->t_org, disc );
220:                                    if(r != n)
221:                                            return -1;
222:                                    t += n;
223:                                    if((s -= n) <= 0)
224:                                            break;
225:                                    a += n;
226:                            }
227:
228:                    next:   if((size -= s) == 0)
229:                                    break;
230:                    }
231:            }
232:    }
```

Sun Aug 14 11:55:45 1994

peregrine.sqo.att.com|/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c

```
233:    }
234:
235:  return 0;
236:  }
237:
238:  #if __STD_C
239:  long vdupdate(Void_t* src, long n_src, Void_t* tar, long n_tar, Vddisc_t* disc)
240:  #else
241:  long vdupdate(src, n_src, tar, n_tar, disc)
242:  Void_t*           src;    /* source string if any */
243:  long              n_src;  /* length of src        */
244:  Void_t*           tar;    /* target space if any  */
245:  long              n_tar;  /* size of tar          */
246:  Vddisc_t* disc;
247:  #endif
248:  {
249:    Table_t tab;
250:    uchar   *data, magic[8];
251:    int     n, r;
252:    long    t, p, window;
253:    Vdio_f  readf, writef;
254:
255:    if(!disc || !disc->readf || (!tar && !disc->writef) )
256:            return -1;
257:    readf = disc->readf;
258:    writef = disc->writef;
259:
260:    /* initialize I/O buffer */
261:    RINIT(&tab.io,disc);
262:
263:    /* check magic header */
264:    data = (uchar*)(VD_MAGIC);
265:    for(n = 0; data[n]; ++n)
266:            ;
267:    if((*_Vdread)(&tab.io,magic,n) != n)
268:            return -1;
269:    for(n -= 1; n >= 0; --n)
270:            if(data[n] != magic[n])
271:                    return -1;
272:
273:    /* get true target size */
274:    if((t = (long)(*_Vdgetu)(&tab.io,0)) < 0 || (tar && n_tar != t) )
275:            return -1;
276:    n_tar = t;
277:
278:    /* get true source size */
279:    if((t = (long)(*_Vdgetu)(&tab.io,0)) < 0 || (src && n_src != t) )
280:            return -1;
281:    n_src = t;
282:
283:    /* get window size */
284:    if((window = (long)(*_Vdgetu)(&tab.io,0)) < 0)
285:            return -1;
286:
287:    tab.compress = n_src == 0 ? 1 : 0;
288:
289:    /* if we have space, it'll be faster to unfold */
290:    tab.tar = tab.src = NIL(uchar*);
291:    tab.t_alloc = tab.s_alloc = 0;
292:
```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com:/n/gryphon/q7/kpv/Software/src/lib/vdelta/vdupdate.c

```
293:    if(n_tar > 0 && !tar && window < (long)MAXINT)
294:            n = (int)window;
295:    else    n = 0;
296:    if(n > 0 && (tab.tar = (uchar*)malloc(n*sizeof(uchar))) )
297:            tab.t_alloc = 1;
298:
299:    if(n_src > 0 && !src && window < (long)MAXINT)
300:            n = (int)window;
301:    else if(n_src == 0 && window < n_tar && !tar && HEADER(window) < (long)MAXINT
)
302:            n = (int)HEADER(window);
303:    else    n = 0;
304:    if(n > 0 && (tab.src = (uchar*)malloc(n*sizeof(uchar))) )
305:            tab.s_alloc = 1;
306:
307:    for(t = 0; t < n_tar; )
308:    {       tab.t_org = t;  /* current location in target stream */
309:
310:            if(n_src == 0)  /* data compression */
311:            {       tab.s_org = 0;
312:
313:                    if(t == 0)
314:                            tab.n_src = 0;
315:                    else
316:                    {       tab.n_src = HEADER(window);
317:                            p = t - tab.n_src;
318:                            if(tar)
319:                                    tab.src = (uchar*)tar + p;
320:                            else if(tab.src)
321:                            {       n = (int)tab.n_src;
322:                                    if(tab.tar)
323:                                    {       data = tab.tar + tab.n_tar - n;
324:                                            memcpy((Void_t*)tab.src,(Void_t*)data
,n);
325:                                    }
326:                                    else
327:                                    {       r = (*readf)(VD_TARGET,tab.src,n,p,di
sc);
328:                                            if(r != n)
329:                                                    goto done;
330:                                    }
331:                            }
332:                    }
333:            }
334:            else    /* data differencing */
335:            {       tab.n_src = window;
336:                    if(t < n_src)
337:                    {       if((t+window) > n_src)
338:                                    p = n_src-window;
339:                            else    p = t;
340:
341:                            tab.s_org = p;
342:
343:                            if(src)
344:                                    tab.src = (uchar*)src + p;
345:                            else if(tab.src)
346:                            {       n = (int)tab.n_src;
347:                                    r = (*readf)(VD_SOURCE,tab.src,n,p,disc);
348:                                    if(r != n)
349:                                            goto done;
```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com:/n/gryphon/q7/kpv/Software/src/lib/vdelta/vdupdate.c

```
350:                            }
351:                    }
352:            }
353:
354:            if(tar)
355:                    tab.tar = (uchar*)tar+t;
356:            tab.n_tar = window < (n_tar-t) ? window : (n_tar-t);
357:
358:            K_INIT(tab.quick,tab.recent,tab.rhere);
359:            if(vdunfold(&tab) < 0)
360:                    goto done;
361:            if(!tar && tab.tar)
362:            {       p = (*writef)(VD_TARGET,(Void_t*)tab.tar,(int)tab.n_tar,t,dis
c);
363:                    if(p != tab.n_tar)
364:                            goto done;
365:            }
366:
367:            t += tab.n_tar;
368:    }
369:
370:  done:
371:    if(tab.t_alloc)
372:            free((Void_t*)tab.tar);
373:    if(tab.s_alloc)
374:            free((Void_t*)tab.src);
375:
376:    return t;
377:  }
```

59 60

Sun Aug 14 11:54:34 1994

peregrine.goo.att.com!/n/cryphon/g7/kpv/Software/src/lib/vdelta/vdio.c

```
  1: #include  "vdelhdr.h"
  2:
  3:
  4: #if __STD_C
  5: static _vdfillbuf(reg Vdio_t* io)
  6: #else
  7: static _vdfillbuf(io)
  8: reg Vdio_t*        io;
  9: #endif
 10: { reg int n;
 11:
 12:   if((n = (*READF(io))(VD_DELTA,BUF(io),BUFSIZE(io),HERE(io),DISC(io))) > 0)
 13:   {       ENDB(io) = (NEXT(io) = BUF(io)) + n;
 14:           HERE(io) += n;
 15:   }
 16:   return n;
 17: }
 18:
 19: #if __STD_C
 20: static _vdflsbuf(reg Vdio_t* io)
 21: #else
 22: static _vdflsbuf(io)
 23: reg Vdio_t*        io;
 24: #endif
 25: { reg int n;
 26:
 27:   if((n = NEXT(io) - BUF(io)) > 0 &&
 28:      (*WRITEF(io))(VD_DELTA,BUF(io),n,HERE(io),DISC(io)) != n)
 29:           return -1;
 30:
 31:   HERE(io) += n;
 32:   NEXT(io) = BUF(io);
 33:   return BUFSIZE(io);
 34: }
 35:
 36: #if __STD_C
 37: static ulong _vdgetu(reg Vdio_t* io, reg ulong v)
 38: #else
 39: static ulong _vdgetu(io,v)
 40: reg Vdio_t*        io;
 41: reg ulong v;
 42: #endif
 43: { reg int          c;
 44:
 45:   for(v -= I_MORE-1;;)
 46:   {       if((c = VDGETC(io)) < 0)
 47:                   return (ulong)(-1L);
 48:           if(!(c&I_MORE) )
 49:                   return ((v<<I_SHIFT) | c);
 50:           v = (v<<I_SHIFT) | (c & (I_MORE-1));
 51:   }
 52: }
 53:
 54: #if __STD_C
 55: static _vdputu(reg Vdio_t* io, ulong v)
 56: #else
 57: static _vdputu(io, v)
 58: reg Vdio_t*        io;
 59: reg ulong v;
```

61 62

Sun Aug 14 11:54:34 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdio.c

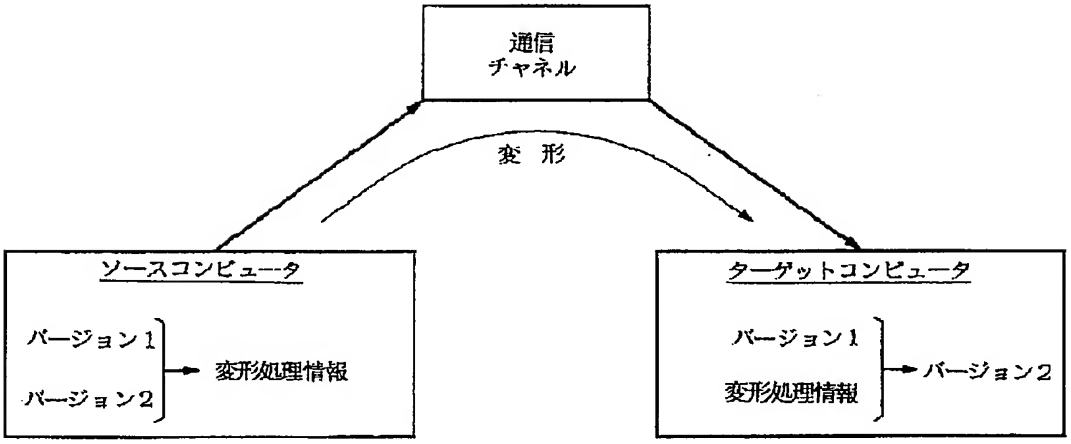
```
 60: #endif
 61: {
 62:   reg uchar       *s, *next;
 63:   reg int         len;
 64:   uchar           c[sizeof(ulong)+1];
 65:
 66:   s = next = &c[sizeof(c)-1];
 67:   *s = I_CODE(v);
 68:   while((v >>= I_SHIFT) )
 69:           *--s = I_CODE(v)|I_MORE;
 70:   len = (next-s) - 1;
 71:
 72:   if(REMAIN(io) < len && _vdflsbuf(io) <= 0)
 73:           return -1;
 74:
 75:   next = io->next;
 76:   switch(len)
 77:   {
 78:   default: memcpy((Void_t*)next,(Void_t*)s,len); next += len; break;
 79:   case 3: *next++ = *s++;
 80:   case 2: *next++ = *s++;
 81:   case 1: *next++ = *s;
 82:   }
 83:   io->next = next;
 84:
 85:   return len;
 86: }
 87:
 88: #if __STD_C
 89: static _vdread(Vdio_t* io, reg uchar* s, reg int n)
 90: #else
 91: static _vdread(io, s, n)
 92: Vdio_t*         io;
 93: reg uchar*      s;
 94: reg int         n;
 95: #endif
 96: {
 97:   reg uchar*      next;
 98:   reg int         r, m;
 99:
100:   for(m = n; m > 0; )
101:   {       if((r = REMAIN(io)) <= 0 && (r = _vdfilbuf(io)) <= 0)
102:                   break;
103:           if(r > m)
104:                   r = m;
105:
106:           next = io->next;
107:           MEMCPY(s,next,r);
108:           io->next = next;
109:
110:           m -= r;
111:   }
112:   return n-m;
113: }
114:
115: #if __STD_C
116: static _vdwrite(Vdio_t* io, reg uchar* s, reg int n)
117: #else
118: static _vdwrite(io, s, n)
119: Vdio_t*         io;
```

Sun Aug 14 11:54:34 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdio.c

```
120: reg uchar*      s;
121: reg int         n;
122: #endif
123: {
124:   reg uchar*      next;
125:   reg int         w, m;
126:
127:   for(m = n; m > 0; )
128:   {       if((w = REMAIN(io)) <= 0 && (w = _vdflsbuf(io)) <= 0)
129:                   break;
130:           if(w > m)
131:                   w = m;
132:
133:           next = io->next;
134:           MEMCPY(next,s,w);
135:           io->next = next;
136:
137:           m -= w;
138:   }
139:   return n-m;
140: }
141:
142:
143: Vdbufio_t _Vdbufio =
144: { _vdfilbuf,
145:   _vdflsbuf,
146:   _vdgetu,
147:   _vdputu,
148:   _vdread,
149:   _vdwrite
150: };
```

【図１】



【図４】

01000100	0
00000010	x y
01000110	20
01000101	9

【図１Ａ】

語 → When buying coconuts, make sure that they are crack free and
位置 → 1 2 3 4 5 6 7 8 9 10 11

have no mold on them. Shake them to make sure that they are
12 13 14 15 16 17 18 19 20 21 22 23 24

heavy with water. Now hold a coconut in one hand over a sink
25 26 27 28 29 30 31 32 33 34 35 36 37

and hit it around the center with the claw end of a hammer.
38 39 40 41 42 43 44 45 46 47 48 49 50

バージョン 1

単語表

1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	
10 free	25 in	
11 and	26 one	
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

【図１Ｂ】

Hawaiian　　all
When buying coconuts, make sure ~~that they~~ are crack free and
1 2 3 4 5 6 7 8 9 10 11

have no mold on them. Shake them to make sure that they are
12 13 14 15 16 17 18 19 20 21 22 23 24

heavy with water. Now hold a coconut in one hand over a sink
25 26 27 28 29 30 31 32 33 34 35 36 37

a brick
and hit it around the center with ~~the claw end of a hammer~~.
38 39 40 41 42 43 44 45 46 47 48 49 50

バージョン 2

【図１Ｃ】

When buying [Hawaiian] coconuts, make sure [all] are crack free and
1 2 3 4 5 6 7 8 9 10 11

have no mold on them. Shake them to make sure that they are
12 13 14 15 16 17 18 19 20 21 22 23 24

heavy with water. Now hold a coconut in one hand over a sink
25 26 27 28 29 30 31 32 33 34 35 36 37

and hit it around the center with [a brick].
38 39 40 41 42 43 44 45 46

バージョン 2

【図１Ｄ】

[When buying] coconuts, make sure that they are crack free and
1 2 3 4 5 6 7 8 9 10 11

have no mold on them. Shake them to make sure that they are
12 13 14 15 16 17 18 19 20 21 22 23 24

heavy with water. Now hold a coconut in one hand over a sink
25 26 27 28 29 30 31 32 33 34 35 36 37

and hit it around the center with the claw end of a hammer.
38 39 40 41 42 43 44 45 46 47 48 49 50

バージョン 1

When buying
1 2

COPY 2 1
命令 1

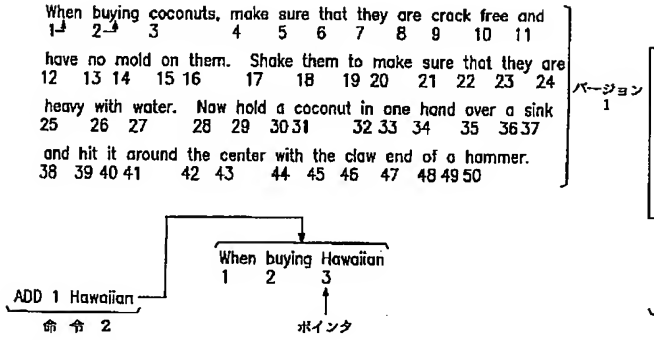
ポインタ

単語表

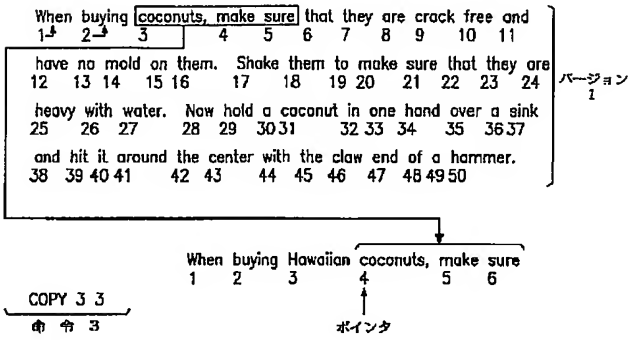
1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	39 Hawaiian
10 free	25 in	40 all
11 and	26 one	41 brick
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

新語

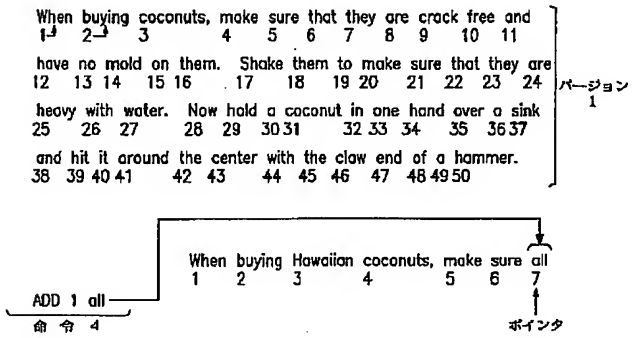
【図１Ｅ】



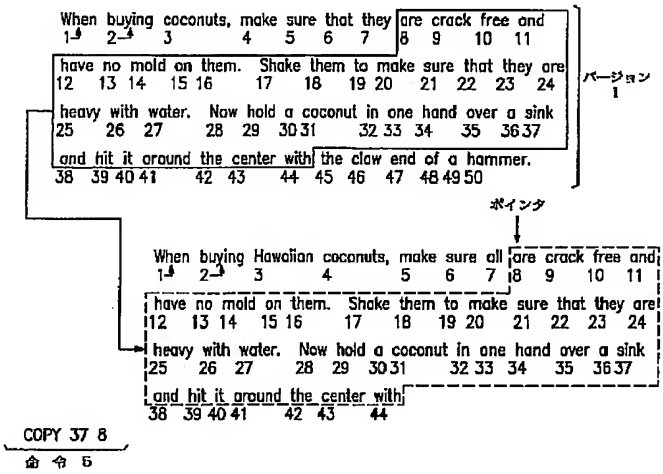
【図１Ｆ】



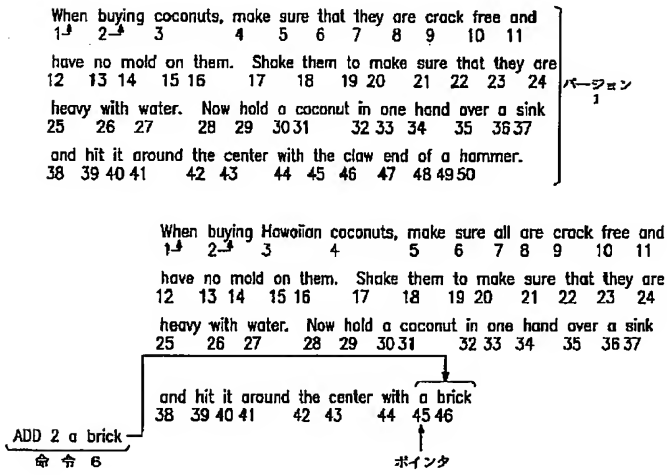
【図１Ｇ】



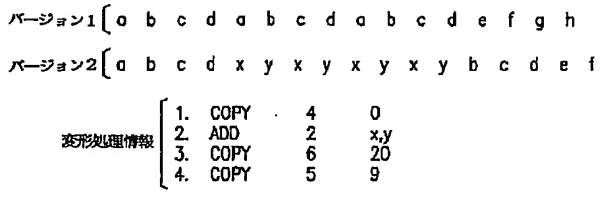
【図１Ｈ】



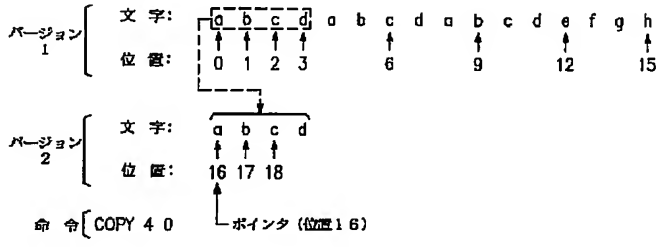
【図１Ｉ】



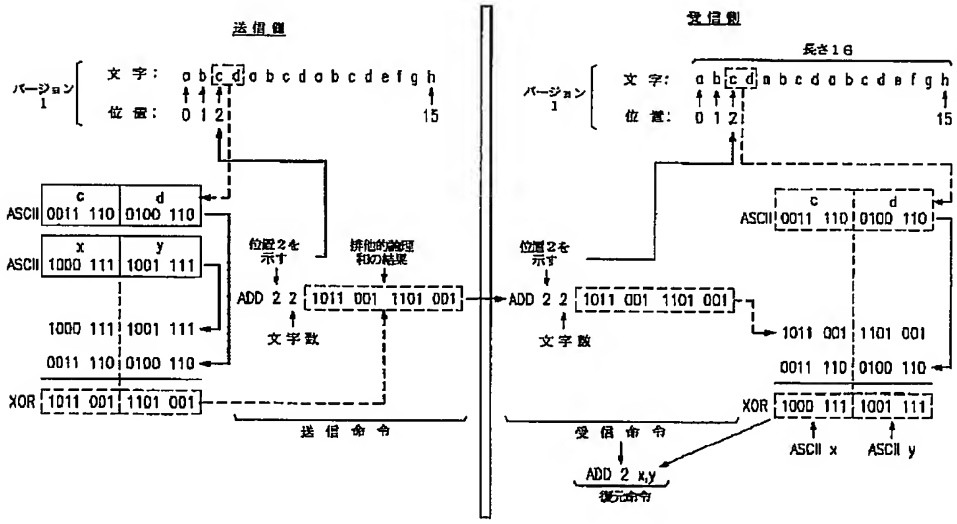
【図２】



【図２Ｂ】



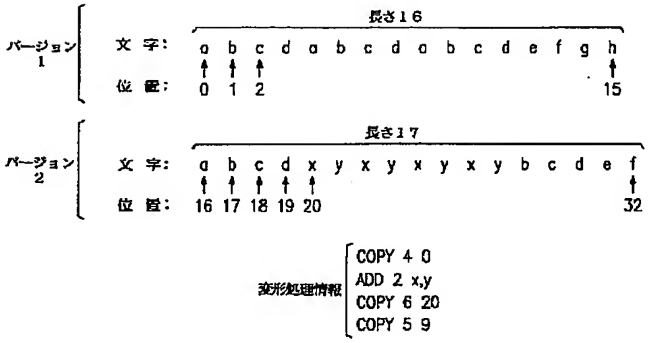
【図１Ｊ】



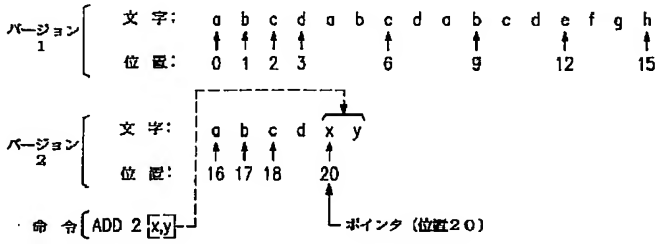
【図６】

```
1.  insert(T,seq,p,orig)
2.  {   key = seq[p,p+1,...,p+MIN-1];
3.      pos = p+orig;
4.      Insert (key,pos) into T;
5.  }
```

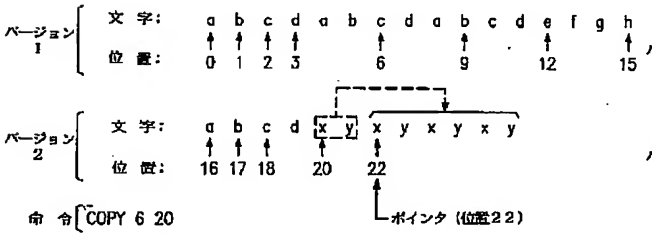
【図２Ａ】



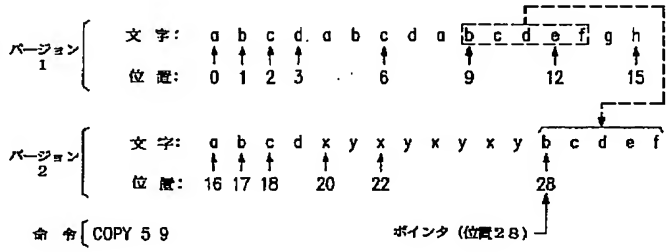
【図２Ｃ】



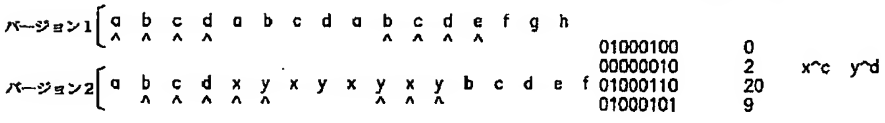
【図２Ｄ】



【図２Ｅ】

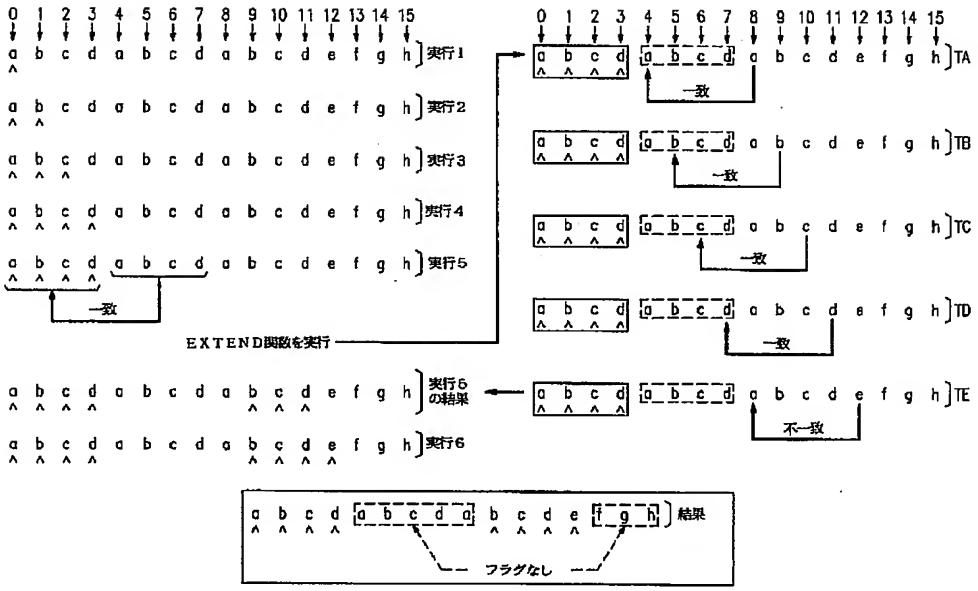


【図９】

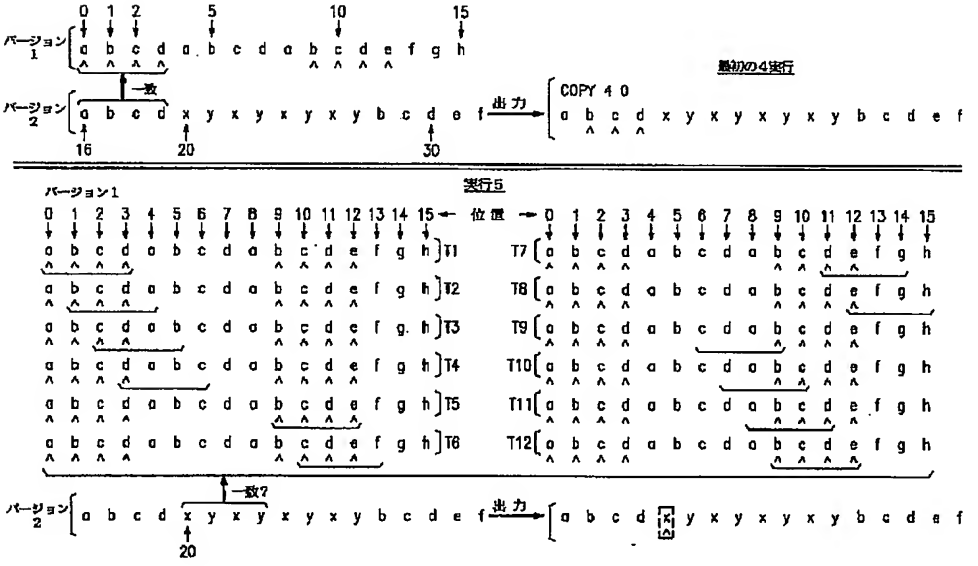


【図１０】

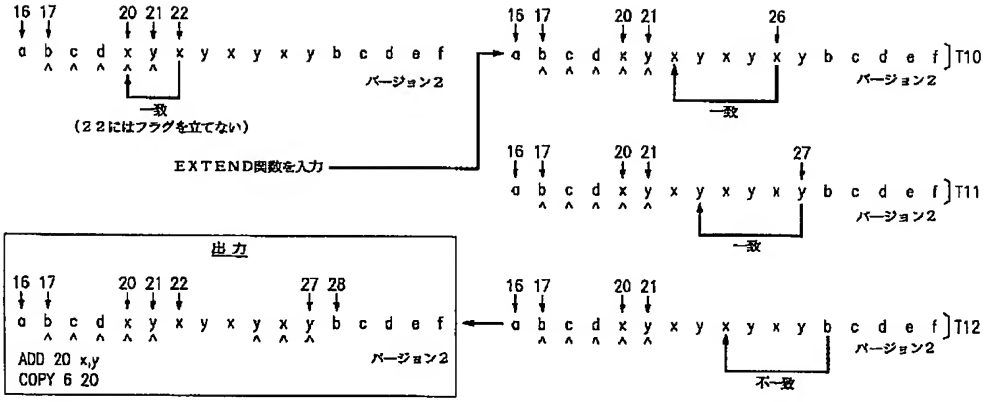
【図２Ｆ】


実行1
実行2
実行3
実行4
実行5
一致
EXTEND関数を実行
実行6の結果
実行6
TA
TB
TC
TD
TE
不一致
結果
フラグなし

【図２Ｇ】


バージョン1
バージョン2
一致
出力
COPY 4 0
最初の4実行
実行5
位置
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
一致?

【図２Ｈ】



【図３】

```
1.  n = length(VERSION1);
2.  C = 0
3.  while(true)
4.  {   inst = readinst();
5.      if(inst == end-of-file)
6.          return;
7.      s = readsize();
8.      if(inst == ADD)
9.          readdata(VERSION2+c,s);
10.     else
11.     {   p = readpos();
12.         if(p < n)
13.             copy(VERSION2+c,VERSION1+p,s);
14.         else copy(VERSION2+c,VERSION2+p-n,s);
15.     }
16.     c = c + s;
17. }
```

【図５】

```
1.  Initialize table of positions T to empty;
2.  process(VERSION1);
3.  process(VERSION2);
4.  process(seq)
5.  {   n = length(VERSION1);
6.      m = length(seq);
7.      c = 0;
8.      add = -1;
9.      pos = -1;
10.     len = MIN-1;
11.     while(1)
12.     {   while(1)
13.         {   p = search(T,seq,c,len);
14.             if(p < 0)
15.                 break;
16.             len = extend(T,seq,c,p,len);
17.             pos = p;
18.         }
19.         if(pos < 0)
20.         {   if(add < 0)
21.                 add = c;
22.             if(seq == VERSION1)
23.                 insert(T,seq,c,0);
24.             else insert(T,seq,c,n);
25.             c = c + 1;
26.         }
27.         else
28.         {   if(seq == VERSION2)
29.                 writeinst(add,c,pos,len);
30.             p = c+len-(MIN-1);
31.             while(p < c+len)
32.             {   if(seq == VERSION1)
33.                     insert(T,seq,p,0);
34.                 else insert(T,seq,p,n);
35.                 p = p+1
36.             }
37.             c = c + len;
38.             add = -1; pos = -1; len = MIN-1;
39.         }
40.         if(c >= m-MIN)
41.             break;
42.     }
43.     if(seq == VERSION2 and add >= 0)
44.         writeinst(add,m,-1,0);
45. }
```

【図８】

```
1.  extend(T,seq,c,p,len)
2.  {   n = length(VERSION1);
3.      if(p < n)
4.          str = VERSION1;
5.      else
6.      {   str = VERSION2;
7.          p = p-n;
8.      }
9.      m = length(seq);  i = c + len + 1;
10.     n = length(str);  j = p + len + 1;
11.     while(i < m and j < n)
12.         if(seq[i] != seq[j])
13.             break;
14.     return i-c;
15. }
```

【図７】

```
1.  search(T,seq,c,len)
2.  {   n = length(VERSION1);
3.      p = c + len − (MIN−1);
4.      key = seq[p,p+1,...,p+MIN−1];
5.      for(each entry e in T that matches key)
6.      {   pos = position(e);
7.          if(p >= n)
8.          {   str = VERSION2;
9.              q = pos−n;
10.         }
11.         else
12.         {   str = VERSION1;
13.             q = pos
14.         }
15.         d = q − (len − (MIN−1) );
16.         if(d >= 0)
17.             if(seq[c,c+1,...,p−1] == str[d,d+1,...,q−1])
18.                 return pos − (len − (MIN−1) );
19.     }
20.     return −1;
21. }
```

フロントページの続き

(72)発明者　カームーフォン　ヴォー
アメリカ合衆国　07922　ニュージャーシィ，バークレイ　ハイツ，スウェンソンサークル　80